Testiranje web aplikacija u alatu Selenium

Kokot, Dražen

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dražen Kokot

TESTIRANJE WEB APLIKACIJA U ALATU SELENIUM

ZAVRŠNI RAD

Varaždin, 2025.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dražen Kokot

JMBAG: 0016145515

Studij: Informacijski sustavi

TESTIRANJE WEB APLIKACIJA U ALATU SELENIUM

ZAVRŠNI RAD

Mentor:

doc. dr. sc. Matija Novak

Varaždin, veljača 2025

Dražen Kokot

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Izrada web aplikacija prolazi kroz nekoliko faza od same ideje do razvoja te krajnje isporuke. Postoje dodatni koraci koji osiguraju web aplikaciju koja radi ispravno i prema specifikacijama korisnika. Jedna od faza za provjeru ispravnosti pojedinih dijelova aplikacije, programskog koda i aplikacije u cjelini su testiranja. Testiranje počinje u samoj implementaciji web aplikacije i nastavlja se nakon završetka kodiranja.

Web aplikacija je web stranica koja sa svojim funkcionalnostima proširuje i nadograđuje statične web stranice. Web stranice pojavljuju se u raznim tipovima od onih namijenjene za kupnju koje su prilagođene korisnicima do statičnih koje su jednake svim korisnicima.

Izgradnja web aplikacija se ostvari kroz uporabu jezika oznaka HTML, kaskadnim jezikom CSS za dodjelu stilova, JavaScript jezikom za dodjelu klijentskih dinamičkih svojstva te PHP za serversku implementacijsku stranu. Tijekom pisanja programskog koda primjerice u PHP-u mogu se pisati prije, tijekom i nakon implementacijske faze jedinični testovi koji provjeravaju ispravnost metoda. Selenium se može koristiti kao sveobuhvatno rješenje za provedbu od jediničnih testova do automatiziranih testiranja programskog koda i web aplikacije.

Ključne riječi: PHP; Testiranje; Selenium; HTML; CSS;

Sadržaj

1. Uvod

Web aplikacije postale su nezaobilazan dio svakodnevnog života i utječu značajno na poduzetnike i pojedince. One omogućuju javno predstavljanje i promidžbu proizvoda i usluga, što je ključno za rast i razvoj poduzeća. Korištenjem web aplikacija, poduzetnici mogu povećati svoj doseg, privući nove klijente i poboljšati poslovne operacije. Poduzeća također saznaju koji proizvodi i usluge su klijentima interesantni, što ujedno pomaže poduzeću u donošenje odluka i strategije. Za pojedinci koji žele unaprijediti svoje znanje iz programiranja mogu koristiti web aplikacije poput W3Schools-a. Web aplikacije također igraju ključnu ulogu u složenim sustavima, koji često dijele funkcije na različite podsustave. Primjerice, u zrakoplovnoj industriji, sustavi za prodaju karata mogu biti realizirani kao web aplikacije, dok se upravljanje zrakoplovima može oslanjati na ugrađene sustave. Ova podjela omogućuje specijaliziranu funkcionalnost i poboljšava efikasnost poslovanja.

Rad sagledava gradivne blokove web stranice i testiranje alatom Selenium. Rad se može podijeliti na dva dijela. U prvom dijelu razrađuje se teorija, koja obuhvaća poglavlja od *2. Web aplikacije* do *5. Selenium*. Drugi dio rada bavi se praktičnom primjenom teorije i sadrži poglavlje *6. Praktičan rad.* Poglavlje nakon uvodnog je *2. Web aplikacije* koje govori o podjeli i diferencijaciji web stranice i web aplikacije. Prvo potpoglavlje bavi se definiranjem vrsta web aplikacija i njihove karakteristike. Veći dio teksta posvećen je dinamičnim i statičnim stranicama, dok se ostatak bavi ostalim vrstama. Drugo i posljednje potpoglavlje govori o tome kako od zahtjeva fazama razvoja nastane gotova web aplikacija. Sljedeće poglavlje *3. Programski jezici* bavi se definicijom, opisom karakteristika i vrsta programskih jezika. Spominju se jezici: HTML, CSS, JavaScript, PHP, SQL koji se koriste u radu, ali je izdvojeno vrijeme za upoznavanje s drugim jezicima poput C#-pa i programskim okvirima. Poglavlje *4. Testiranje* zaduženo je za upoznavanje s procesom i vrstama testiranja. Izrađene su dvije velike podjele na funkcionalna i ne funkcionalna testiranja. Podjele testiranja se dalje dijele na specifična testiranja koja su zadužene za aspekte aplikacije. Primjerice za aspekt sigurnosti provodi se testiranja sigurnosti. Posljednje poglavlje *5. Selenium* opisuje odabrani alat kojim se u praktično dijelu testira web aplikacija. Drugi dio rada bavi se razradom praktično djela. U praktičnom dijelu opisuje se sam proces kojim se od zahtjeva razvila web aplikacija i provodilo testiranje.

2. Web aplikacija

Kako se rad bavi testiranjem web aplikacije potrebno je razjasniti što je uopće web aplikacija. Web stranica, web aplikacija i web rješenje su usko povezani pojmovi i često se koriste kao sinonimi no postoji bitna razlika između spomenutim. Prema [1] postoje pojmovi: web stranica, web aplikacija i web rješenja. U suštini pojmovi se razlikuju po razini kompleksnosti potrebnoj za realizaciju korisničkih zahtjeva. Ovisno o korisničkim zahtjevima programer odabire hoće li realizirati rješenje u obliku web stranice, web aplikacije ili web rješenja.

Web stranica može se smatrati kao osnovni gradivni element koji se nadograđivanjem poput programske logike, dodatnim funkcionalnostima i serverskim jezikom se obogati i pretvori u web aplikaciju. Web stranica nije ništa drugo nego dokument s HTML kodom koji se može prikazati u web pregledniku. Sadržaj se ne mijenja od više korisnika već je jedan korisnik donosi sve izmjene bio to administrator ili vlasnik.

Nadogradnjom web stranice korištenjem primjerice PHP skriptnog jezika stranica prelazi u web rješenje. Najbitniji faktor tranzicije su funkcionalnosti koje web rješenje omogućuju korisnicima i manipulacija sadržaja. Korisnik ima puno veću ulogu na rad web aplikacije i utjecaj na sadržaj nego kod web stranice. Umjesto da korisnik bude samo promatrač on može ako ima ovlasti mijenjati sadržaj, brisati i stvarati novi. Korištenje baze podataka koja služi kao spremište i izvor podataka korišteni u web aplikaciji. Korisnici kroz ovlasti dobivaju prava pristupa bazi i funkcionalnostima. Veći broj korisnika donosi promjene i u interakciji je s web aplikacijom nego kod web stranice. Web aplikacije mogu biti primarno realizirane kao statične i dinamične više u sljedećem poglavlju gdje se detaljnije objašnjavaju tipovi web aplikacija.

Prema izvorima [2], [3] složeni sustavi koji su dio poslovnog sustava i prisutni u gotovo svim aspektima poslovanja od prodaje do skladištenja mogu biti realizirani web tehnologijama. Tako složeni sustav u interakciji je s dugim sustavima i okruženjima. Razina kompleksnosti prema obuhvatu alata, tehnika, platforma i programskih jezika je veća od web aplikacije. U ovom radu definirani je pojam 'web rješenje' kao sustav koji omogućuje interakciju između različitih poslovnih aspekata koristeći web tehnologije. Primjeri web rješenja su: CRM(Customer Relationship Management) i ERP(Enterprise Resource Planning) sustavi.

Web stranice su primarno informativnog karaktera u obliku brošura gdje je sav sadržaj izgrađen od jedne osobe. Web aplikacije imaju interaktivnu karakteristiku gdje korisnici stvaraju i mijenjaju sadržaj. Web rješenja je ljestvica iznad web aplikacije i ona je rješenje za komunikaciju unutar poslovnog sustava.

2.1. Tipovi web aplikacija

Aplikacije se izrađuju pomoćnu velikog broja alata i tehnika kojima se realiziraju funkcionalnosti zahtijevane od korisnika. Korisnik specifikacijom zahtjeva diktira koliko će rješenje biti složeno, detaljizirano te koliko će se funkcije pružati korisnicima. Rješenje se može podijeliti u jedan od dva primarna tipova web aplikacije. Postoji brojne podjele web aplikacija prema [4] ih primarno možemo podijeliti na statične i dinamične. Postoje i drugi tipovi web aplikacija poput jednostranične, višestranične, animirane, portali, web trgovine i sustavi za upravljanje sadržajem koji se mogu smjestiti u primarne tipove web aplikacija.

Kako su primarni tipovi podijeljeni prema interakciji korisnika sa stranicom, prvo će biti objašnjene statične web aplikacije. Web aplikacije koje imaju karakteristiku isključivo prikaza informacija i sadržaja koji se ne mijenja tj. rijetko će se mijenjati. Statične web aplikacije mogu su izraditi samo jednim dokumentom, ali i povezivanjem dokumenata. Dokument je statičnog karaktera i spremljen je nastavkom .html jer svakom korisniku prikazuje isti sadržaj bez obzira na lokaciju, jezik ili. drugi karakter. Dokument je pisan u jeziku oznaka HTML gdje se koriste tagovi za strukturiranje sadržaja. Stilskim jezikom CSS dodaju se i vizualno mijenjaju postojeći i novo izrađeni dijelovi. Korisnici nisu u interakciji s web aplikacijom u smislu promjena postojećeg sadržaja, stvaranje novog ili brisanje sadržaja. Zbog toga što je sadržaj i količina informacija vrlo malen postavlja se izravno u .html dokument umjesto da se sprema u bazu podataka. Kao primjer statičnih web aplikacija izdvajaju se portfelji koji su izrađeni za konkretnu osobu i služi kao mjesto prikaza svih dosadašnjih projekata.

Suprotnost statičnim aplikacijama predstavljaju dinamične koje se razlikuju po aspektnu korisničkog utjecaja na sadržaj i ponašanje same web aplikacije. Veći je broj informacija i autora koji stvaraju informacije. Umjesto da sadržaj bude rijetko kad izložen promjenama kod dinamičnih web aplikacija sadržaj se stalno mijenja, nadodaje i briše. Korisnici su kontrolirani u kojoj mjeri utječu na sadržaj korištenjem uloga. Generalno uloge se dijele na četi kategorija koje su: neregistrirani, registrirani, moderator i admin, prema ovom redoslijedu uloga neregistrirani korisnik ima najmanji utjecaj i smanjene funkcionalnosti web aplikacije dok admin im najveće. Veliki dio sadržaja je promjenjiv poput: tekst u člancima, slike, naslovi zbog čega se javlja potreba za bazom podataka. Jako mali dio sadržaja će biti statičan primjerice: stavke glavnog izbornika, početna stranica koje nije potrebno izdvojiti u bazu.

Primjer dinamične web aplikacije je blog u kojemu se može vidjeti koliki utjecaj na sadržaj ima korisnik s pripadajućom ulogom. Korisnik koji posjećuje web aplikaciju pripada u ulogu neregistrirani korisnik zbog toga ne može dati osvrt na pročitani članak u obliku komentara dok registrirani može. Uloga moderator za razliku od registriranih korisnika može upravljati člancima drugih korisnika dok registrirani mogu mijenjati samo svoje. Posljednja

uloga s najvećim brojem funkcija su administratori koji imaju sve mogućnosti prije spomenutih uloga no mogu i upravljati ulogama niže razine primjerice obrisati korisnike tipa moderator.

Javljaju se dodatni tipovi web aplikacija koji se znatno razlikuju po ponašanju i namjeni u odnosu na statične i dinamične web aplikacije. SPA web aplikacije (*eng. Single Page Applications*) su aplikacije gdje je dio statičan, a drugi dio dinamičan. Zbog kombinacije statičnog i dinamičnog aspekta SPA web aplikacije pripadaju u zasebnu kategoriju. Statični dijelovi poput zaglavlja i podnožja ne zahtjvaju promjene dok se dinamičan dio poput inboxa stalno ažurira promjenama. Prema [5] umjesto osvježenja cijele web stranice osvježiti će se samo čvor DOM-a. Zbog toga korisnik ne mora osvježiti stranicu kako bi vidio promjene već se one generiraju i prikazuju gotovo instantno. SPA se koristi u programksim okvirima: ReactJS Angular i Blazor. Primjer SPA web aplikacije je Google mail gdje su statični dijelovi navigacija, a inbox sadrži prošle i pristigle poruke.

Posljednje web aplikacije koje omogućuju korisniku znatan utjecaj na sadržaj web aplikacije su CMS-i (eng. Content Management System). Sadržaj se uređuje vizualno i ne zahtijeva od korisnika programsko iskustvo. CMS-i poput WordPress-a dodatcima omogućuju korisniku izgradnju cijele web aplikacije poput izrade: navigacije, sekcija i podnožja.Postoje i dodatne podjele web aplikacije na daljnje kategorije no za ovaj rad sagledani su najrasprostranjeniji tipovi web aplikacija. Neki od dodatnih tipova su: animirane, progresivne, portal i RIAs (eng. Rich Internet Applications) web aplikacije.

2.2. Faze razvoja

Web aplikacija od same ideje do isporučenog proizvoda prolazi kroz faze razvoja. Postoje razni pristupi koji se koriste za razvoj web aplikacija no prema [6] svi koriste iste faze razvoja, a to su: specifikacija korisničkih zahtjeva, dizajn, razvoj, testiranje i održavanje. Faza razvoja može se definirati kao cjelina zadužena za rješavanje jednog od životnih ciklusa web aplikacije. Ovisno o pristupu neka od faza razvoja mogu biti grupirani u isti proces, ali i odvojena u zasebni proces. Prilikom razvoja web aplikacije svaka od njih se razlikuje prema korisničkim zahtjevima što ujedno utječe na faze razvoja. U jednoj web aplikaciji bit će izražena faza razvoja specifikacija korisničkih zbog složenost zahtjeva i domena dok za jednostavniju web aplikaciju ta faza neće toliko zahtijevati. Svaka faza doprinosi razvoju funkcionalne web aplikacije i pomaže timu podjelu rada.

Specifikacija korisničkih zahtjeva je proces definiranja i analiziranje korisničkih zahtjeva te dokumentiranje istih. Zahtjevi se mogu podijeliti na funkcionalne koji specificiraju što se zahtjeva da aplikacija radi i ne funkcionalni poput: performansa, sigurnosti te korisničkog

iskustva. Proces definiranja zahtjeva odvija se između klijenta ili reprezentativca klijenta i razvojnog tima. Razvojni tim je poduzeće koje se bavi razvitkom programskog rješenja.

Dizajn služi kao razrađena ideja i početni koncept koji se koristi u daljnjim fazama. Korisničko sučelje razrađuju se kroz Wireframe te se stvaraju početni prototipovi. Iz dostupnih podataka dizajnira se baza podataka u obliku dijagrama kod kojeg se definiraju: relacije, veze i atributi. Dodatni dijagrami se izrađuju kojima se bolje razumije i dokumentira web aplikacija. Dijagrami arhitekture koji prikazuje komponente i veze između njih. Kao pomoć u fazi razvoja stvaraju se dijagrami klase gdje se definiraju klase s atributima i metodama. Dijagrami slučaja korištenja prikazuju funkcionalnosti i kako ih korisnici koriste. Podaci kao ključan aspekt web aplikacije prikazuju se u dijagramu toka podataka.

Razvoj je faza zadužena za samu implementaciju i razradu faze dizajna. Programeri koristeći programske jezike i okruženja izrađuju front-end i back-end komponente aplikacije. Programeri u front endu koriste HTML, CSS i JS za izradu djela aplikacije koja nije povezana s bazom podataka. U front end dijelu programeri koriste PHP za povezivanje s bazom podataka. Baza podataka se povezuje s aplikacijom i implementiraju se zahtijevane funkcionalnosti.

Testiraju se pojedine funkcionalnosti kako bi se provjerila ispravnost i zadovoljili predefinirani zahtjevi. Napisani kod testira se kako bi se osigurao kod bez grešaka. Uz testiranje same aplikacije testira se interakcija web aplikacije s web preglednicima i osiguranje da se mogu izvršiti sve funkcionalnosti kroz web preglednike. Nakon uspješnih testiranja isporučuje se web aplikacija s svim potrebnim izvršnim paketima, te se upozna korisnika s radom aplikacije.

Održavanje je posljednja faza u životnom ciklusu web aplikacije, a odnosi se na period nakon isporučenja aplikacije i web aplikacija je postala operativna. U fazi održavanja sve greške se bilježe s obzirom na stupanj ugrožavanja aplikacije. Zabilježene greške se repliciraju i pokušaju riješiti popravkom koda ili otkrivanjem uzroka poput: nedostatka datoteke. Prate se novi trendovi sigurnosti i zastarjeli pristupi i kod se uklanja i zamjenjuje. Podrška korisnicima se pruža oko rješavanja pitanja i briga.

Integracijom i provedbom svih faza dokraja omogućava razvoj web aplikacije koja radi u skladu s korisničkim zahtjevima. Razvojni tim smanjuje greške praćenjem faza i provedbom istih u cijelosti. Korisnik smanjuje troškove jer web aplikacija radi ispravno i ne mora se provoditi daljnje financiranje za ispravke grešaka ili plaćanje neimplementiranih funkcionalnosti.

3. Programski jezici

Prema [7] programske jezik se definira kao jezik koji podržava instrukcije koje kad napisane u računalu mogu biti izvršene. Programske jezici mogu se podijeliti na jezike više i niže razine gdje se jezici više razine pretvore u niže koje računalo direktno izvršava. Postoji veliki niz programskih jezika no gotovo svakog možemo svrstati u jednu od kategorija: machine and assembly, algoritamske, poslovno orijentirane, edukacijsko orijentirane, objektno orijentirane, deklarativne, skriptne, za uređenje dokumenata i WWW jezike prikaza.

Machine jezik sastoji se od sintakse kod koje je sve napisano numeričkim brojevima 0 i 1. Ljestvica iznad machine jezika je assembly kod kojeg se koriste mnemonični kodovi u sintaksi za izvršavanje naredba. Algoritamski jezici poput Fortran-a i Algol-a se koriste za prikazivanje matematičkih izraza i formula. Poslovno orijentirani jezici poput Cobol-ta i SQL se koriste za transformaciju polsovnog aspekta poduzeća u razumljivo računalu u obliku poadatak smještenih u baze podataka. Jezici poput Basic i Logo zbog jednostavne sintakse su korišteni u obrazovne svrhe. Definiranje složenih vrsta podataka, ponovnu iskoristivost i jedna odgovornost funkcije realizirana je kroz objektne jezike poput: C++, C# i Java. Korištenjem deklerativnih jeizika programeru se omogućuje pisanje programa na način da specificira što raditi bez da definira kako primjerice u Prolog-u. WWW jeziici su jezici kojima se sadržaj strukturira i prikazuje u web pretraživaču korištenjem HTML-a i XML-a i vizualno uredi CSS-om. Iako su skriptni jezici poput PHP-a i JS-a namjenjeni za manje probleme mogu se koristiti za složenije probleme. U radu veća pažnja je posvećena skriptnim i WWW jezicima prikaza.

3.1. HTML

HTML je kratica za Hypertext Markup Language. Jezik HTML je jezik oznaka i pripada u WWW jezike prikaza jer obogaćuje sadržaj poput: teksta, slika, videa i zvuka strukturom. [8] Elementi su posebni specifikatori koji signaliziraju web pregledniku koji tip sadržaja je u pitanju i omogućuju pregledniku za obradu sadržaja. Web preglednik diferencira elemente od običnog teksta šiljastim zagradama. Svaki element obgrljen je šiljastim zagradama na početku je znak manje, a nakon ključne riječi taga nalazi se znak veće. Hijerarhijski možemo podijeliti elemente na tagove koji imaju samo početni tag i tagovi koji imaju početni i završni tag. Elementi bez završnih tagova su prazni elementi poput br i hr taga, dok tagovi s početnim i završnim su p, div, head, body i puno drugih tagova. Tagovi poput: p, para, span možemo svrstati u tekstualne tagove. Dodatno tagovi se koriste za vizualno identificiranje naslova, teksta i citate. Tag zadužen za obradu slika je img tag dok video, audio i source su zaduženi za video i audio sadržaj.

Atributi su opcije kojima elementi poprimaju dodatna ponašanja. Atribut se sastoji od imena i vrijednosti atributa. Primjerice tagu img atributom src dodaje se opcije specifikacije putanje slike. Putanje su izvori sadržaja i mogu pored slika koristiti i za video i audio sadržaje. Putanje do izvora mogu biti specificirane relativno s obzirom na projekt, apsolutna lokacija datoteke i kao URL s web mjesta. Iako svaki atribut poprima specifične atribute, zajednički atribute gotovo svim tagovima su id, class i lang.

HTML pojavljuje se u 90-im godinama dvadesetog stoljeća i predstavlja osnovicu za gotovo sve postojeće web stranice. [9] Unutar Europskog nuklearno razvojnog centra CERN pojavljuje se potreba za organizacijom znanstvenih informacija. Iz početne ideje kroz angažman Sir Tim Burns Lee razvio se HTML 1993 godine. Prije inicijalne verzije Sir Tim Burns Lee jer već 1992 razvio prototip. HTML se razvijao u nekoliko verzija i svakom verzijom donosi nove funkcionalnosti od HTML 1.0 do HTML 4.0 dogodile su se brojne promjene. [10] HTML 5.0 je najnovija verzija i standard za izradu web stranica. Početno verzija HTML 1.0 sadržala je vrlo mali broj tagova u usporedbi sa sadašnjom verzijom HTML-a poput: paragrafa, lista, i slika. Uvođenjem HTML 2.0 jezik postane detaljizirani uvođenjem novih tagova. Umjesto slijedne HTML 3.0 proizvođači savjetuju W3C World Wide Web Consortium i objavljena je HTML 3.0 verzija. Verzijom HTML 4.0 uvođeni su dodatni elementi grupacije poput zaglavlja i podnožja tablica objavljena je 1999. Posljednja i najnovija verzija za vrijeme izrade ovog rada je HTML 5.0 koja uvodi semantičke elemente i podržava multimedijske sadržaje. Semantički elementi su elementi konkretnije od prijašnjih verzija opisuju sadržaj i programeru olakšaju pisanje koda kroz jasniju sintaksu. Primjerice uvođeni su tagovi: article, aside, nav, summary i mnogi drugi.

3.2. CSS

CSS skraćenica Cascading Style Sheet je web jezik kojim se na web stranici utječe na vizualni identitet. Utjecaj CSS-a je rasprostranjen na promjenu boja, veličine teksta do izrada jednostavnih animacija. CSS se javlja kao potreba za dodaju vizualnog identiteta jednostavnim HTML stranicama. Promjene se izvršavaju nad HTML elementima. CSS kod koji mijenja vizualni identitet može biti ugniježđeni u sami tag, unutar web stranice unutar script taga te kao vanjskoj datoteci. U slučaju kombinacije više od jednog ili sva tri pristupa sljedećim redoslijedom će kod biti vidljiv: unutar taga, unutar web stranice i na kraju u vanjski datoteci. Svaki od pristupa mora se referencirati na element pošto je ugniježđeni CSS pisan unutar samog HTML elementa ne mora koristiti selektore kao što drugi dva pristupi moraju. CSS pisan u web stranici smješten u style tagu može koristiti unutar stranice u kojoj je definiran.

Napisani CSS u vanjskoj datoteci ima prednost što se datoteka može povezati s ostalim web stranicama i biti im dostupan.

Selektori su identifikatori koji se koriste za povezivanje elementa s CSS stilom. Selektori se mogu podijeliti u tri dominantne vrste: element selektori, id-evi i klase. Selektor tipa element svi stilovi za element nasljeđuju se na ostale stilove. Kad se stil odnosi na p tag bit će naslijeđeni za sve p tagove koji se koriste u web stranici uz uvjet da stranica koristi definirani stil. Korištenjem id-a i klasa može se definirat koji će tagovi posjedovati stil postavljanjem atributa id ili class s vrijednošću istom kao u stilskom bloku. Id i klase razlikuju se po obuhvatu tagova. Klase su općenitije i mogu se odnositi na veći broj tagova na različitim stranicama, a ID-evi se koriste za stiliziranje jednog elementa.

Elementi zauzimaju prostor koji se sagledava kroz box-model. Element posjeduje širinu i visinu koja se definira mjerama poput: pixela, postotaka, milimetri i druga mjerila. Sadržaj je obgrljene sljedećim slojevima iznutra prema van: ispuna, obrub i margina. Svaki od slojeva može se promijeniti upotrebom CSS stilova. Ispuna poprima svojstva kao što su boja pozadine i veličina ispune u neki od smjerova: gore, desno, dolje, lijevo. Obrubi svojstvima širine, boje, i sjene mijenjaju izgled. Margine pozicioniraju element ovisno na roditeljski element. Posljednje svojstvo od značaja je svojstvo pozicija koja može biti: statično, relativno, fiksno, apsolutno i ljepljivo. Statičnu poziciju općenito poprimaju svi elementi. Relativna pozicija odnosi se na uobičajenu poziciju elementa, dok se fiksna pozicija odnosi na body tag. Posljednja pozicija ljepljivo ovisi o poziciji miša.

3.3. JS

Jedan od skriptnih jezika kojem web aplikacija postane dinamična i proširuje funkcionalnosti je JavaScript. Kod programiranja koriste se podaci koji se kroz operacije stvore, promjene, spreme ili obrišu. Kako bi programski jezik koristio podatke iz web pretraživač koristi sučelje DOM eng. Document Object Module. [11] Za razliku od skriptnog jezika PHP koji može prikupiti podatke iz baze, JavaScript prikuplja podatke koristeći DOM-a iz web stranice. U datotečnom modelu objekt je web stranica i njezini dijelovi slike, obrazci i linkovi su objekti sa svojstvima i metodama.

3.4. PHP

PHP je skriptni jezik kojim se može dodati web aplikacije funkcionalnosti poput: spremanje podataka u bazu, prijava i registracija, pamćenje podataka sesijama. PHP je inicijalno bila kraćenica za Personal Homepage no kroz godine skraćenica je promijenila

značenje i stoji kao skraćenica za Hypertext Preprocessor. [12] PHP podržava varijable i kontrolne strukture. Napisani PHP kod kroz PHP interpreter prijevodi u HTML kod kojeg web pretraživač koristi za izradu web aplikacije. Varijable u PHP jeziku varijable započinju znakom dolar kojim se razlikuju od ostalih dijelova koda. Kod sagledavanja tipova varijable iako podržana su podržani tipovi kroz eksplicitnu deklaraciju funkcije strict_types. Varijable se u PHP interpreteru renderiraju kao tekstualni tipovi, zbog toga PHP se može definirati kao loosely skriptni jezik.

Prilikom rada s HTML formama tj. obrascima razlikujemo način slanja podataka metodama post i get. Prilikom slanja get metodom podaci se šalju direktno unutar URL-a vidljivo korisniku. S druge strane post metodom podaci se ne šalju URL-om nego kao dio tijela HTTP zahtjeva. Odabir metode ovisi o vrsti i osjetljivost podataka koji se šalju. Metoda get namijenjena je za jednostavne podatke poput pojma koji se pretražuje u tražilici. Dok je metoda post namijenjena za osjetljivije i veće podatke poput login podataka, slike, bankovne podatke i slične. Dodatna prednost post metode je veća veličina za slanje podataka primjerice za slanje slika iz obrasca.

3.5. Ostalo

Prilikom izrade web stranica mogu se koristiti brojni niz postojećih jezika. Odabir jezika ovisi o korisničkim zahtjevima. Web stranica namijenjena za online predstavljanje poduzeća razlikuje se od web trgovine gdje korisnici kupuju proizvode i zbog toga će se razlikovati jezici i alati korišteni za izradu. Za izradu složenijih stranica javljat će se potreba za bazom podataka i time jezici koji njime upravljaju i povezuju se stranicom.

Prvi od jezika za izradu web stranica je Python koji je objektno orijentirani jezik visoke razine namijenjen za razna programska rješenja od desktop i web rješenja do primjene u rudarenju podataka. U slučaju korištenja Python jezika za izradu web stranica pažnja se stavlja serversku obradu podataka iz baze podatka i rukovanje HTTP zahtjevima i odgovorima. Neke od prednosti korištenja Python-a su: lako razumijevanje sintakse, sofisticirane biblioteke i razvojna okruženja za razvijanje složenih aplikacija. [13]

Jedan od značajnih jezika za razvijanje kompletnih i složenih web rješenja je C#. Jezici C i C++ su utjecali na stvaranje modernog C# jezika s ciljem razvoja aplikacije iz različitih domena. [14] Neke od primjena C# su izrada desktop aplikacija i video igara. U sferi interneta C# kombinira klijentsku i serversku stranu i stvara web aplikacije. Razvojna okruženja poput ADO.NET i .NET MAUI se mogu koristiti za izradu web aplikacije koja je samostalna ili dio većeg rješenja. Blazor je spoj serverske logike i klijentske točnije mogu se izraditi aplikacije bazirane na .NET tehnologiji i izvršavati u web pregledniku. [15]

4. Testiranje

U domeni programskog razvoja testiranje se definira kao provjera ispravnosti koda i provjere usklađenosti napravljenih funkcijskih zahtjeva sa zadanim korisničkim zahtjevima. Prema definiciji kod testiranja izdvajaju se ciljevi koji se žele postići, a to su: dokazivanje kupcu da program zadovoljava zahtjeve i otkrivanje mana. Validacija je upravo rezultat zadovoljavanje korisničkih zahtjeva, dok verifikacija služi kao provjera ispravnosti koda i otkrivanje mana. Ukratko validacija se usredotoči na izradu pravog rješenja, a validacija da rješenje bude izrađeno pravilno. [3]

Zadovoljavanje razine ispunjenosti verifikacije i validacije uveliko ovisi o: očekivanjima za rješenje i novcu. Za rješenje koje mora raditi bez greške gdje su posljedice greški kritične poput sustava za upravljanje avionima ili pumpa za inzulin, testiranje mora biti što cjelovito i detaljnije prevedeno. Rješenja koja nisu kompleksna i mogu u početnoj verziji raditi s greškama imaju smanjenu razinu cijelosti i detaljnosti testiranja poput videoigre. Samim time kompleksnija rješenja će biti skuplja nego manje kompleksna. Za rješenja koja počinju s manje detaljiziranim testovima mogu kroz vrijeme na temelju povratnih korisničkih informacija dodatno razviti i nadograditi. Dodatan pristup koji se može koristiti pored testiranja je inspekcija koja dodatno pomaže u otkrivanju mana. Inspekcija je statičan pristup otkrivanja mana korištenjem znanje iz domene za koje se razvija rješenje, programski jezici i znanje o sustavu. Inspekcijom se može pronaći većina mana, sustav se može sagledati u cijelosti, ali i do sad napravljenom izdanju te šire sagledavanje rješenja u području: zadovoljavanje standarda, održavanje i ažuriranje.

Testiranje i otklanjanje pogrešaka su razdvojeni pojmovi koji označavaju dva različita procesa. Dok testiranje ima doticaj sa samim programskim kodom tako da programski ukazuje na greške u kodu. Otklanjanje pogrešaka se koristi u uklanjanju grešaka u kodu i time je potproces testiranja. Prema [16] testiranje u povijesnom smislu se može podijeliti na sljedeće epohe: otklanjanje pogrešaka, demonstracije, destrukcije, evaluacije, i prevencije. Nadalje u istom radu se navodi razlika između otklanjanje pogrešaka i testiranja. U 50-im godinama 20. stoljeća Charles Baker u osvrtu na knjigu Digital Computer Programming razlikuje otklanjanje pogrešaka i testiranje. Prema Charles Baker-u [16] otklanjanje pogrešaka (eng. debuging) je osiguranje da program radi, dok testiranje osigura da program riješi problem. Epohe nisu isključivo počele objavom jednog rada, već su postojale prije objave rada. Početna epoha otlkanjanje pogrešaka počinje od same pojave programiranja i traje do 1956 godine. Otlkanjanje pogrešaka i testiranje se koristilo kao sinonim i nije bila definirana jasna razlika. Slijedi epoha demonstracije koja traje do 1978 godine za koju je veća pažnja stavljena na testiranje zbog želje smanjenja broja greški u programa. U epohi destrukciji testiranje pokuša

pronaći greške u testiranju i time. Epoha destrukcije je značajna po tome što želi pronaći greške, a ne potvrditi ispravnost sustava i traje do 1987. Evaluacija je epoha slijedi FIPS metodologiju koja ima karakteristiku da koristi prošle testove u sljedećim testovima te traje o 1988. Prevencijska epoha koristi STEP metodologiju razvijenu 1985 od IEEE organizacije. Prema STEP metodologiji testiranje je integrirano u svaki dio životnog ciklusa programskog rješenja. Testiranja se provedu: prije, tijekom i nakon implementacije čak i tijekom promjena se vrši testiranje.

Općenito testiranja se mogu prevoditi kao tri faze tijekom izrade programskog rješenja. Ovisno o korištenom pristupu za razvoj programskog rješenja faze mogu biti spojene, sekvencijalne i ponovno korištene za svaki funkcijski zahtjev. Faze testiranje su: testiranja tijekom implementacije, testiranje izdanja i korisničko testiranje. Svaka od faza zadužena je za testiranje pojedinog aspekta rješenja od samog koda do krajnjih korisnika. Detaljnost faze točnije dubina i širina koliko je razrađena faza ovisi o projektu. Mogu se izdvojiti: programeri, dizajneri, testeri i korisnici koji doprinose testnim fazama.

Tester je osoba zadužena za pisanje, svaka osoba drugačije interpretira i razmišlja o programskom rješenju. Prema [17] važno je da tester ima objektivan i pravi stav u radu testova. Stav tester ne bi trebao biti: dokazati ispravnost napisanog koda ili dokazivanje ispunjenosti zahtjeva. Ispravan stav i cilj testera je pronalaženje što više grešaka u programskog kodu. Naime testiranje vođenju ciljem dokazivanja da god nema greške, rezultira zanemarivanje potencijalnih greški. Test koji ukazuje da programski kod ima grešku i ne izvrši se je uspješan dok testovi bez greške nisu uspješni jer nisu doprinijeli razvoju boljeg rješenja. Strategije pristupa testiranju su: crna i bijela kutija s ciljem maksimiziranje pouzdanosti programskog rješenja. Strategija crne kutije provjerava da unosi zadovoljavaju zahtjeve, vrlo je iscrpno testirati svaki input pa se zbog toga izrade testovi za slične unose. Strategija bijele kutije sagledava interakciju komponenata sustava i izrađuje testove za njih. Obe strategije su nepotpune i zbog toga je bolje da tester slijedi načela testiranja.

Postoje razni pristupi testiranju jedni od njih su: dinamični i statični. Dinamično je kad se testira od samog početka izrade sustava drugim riječima razvoj programskom sustava vođeno testiranjem. Dinamično testiranje odmah rješava iskrsnule greške i čini ih nepostojećim u krajnjoj verziji sustava. Statičko testiranje zasniva se na analizi koda. Analiziraju se uobičajene prakse, algoritmi i česte programske greške. S obzirom na razne pristupe razvoju rješenja javljaju se načela koja se mogu promijeniti u pristupima testiranja. Ističu se sedam načela koji pomažu u otkrivanju mana kroz testiranje. [18] Prvo načelo je testiranje ukazuje na mane, a ne njihovo odsutnost. Za skup ulaza transformacijom se dobiva skup izlaza. Neki od izlaza mogu biti greška zbog krivih ulaza ili transformacije. Za rješenje gotovo je nemoguće provoditi testiranje za sve moguće ulaze i transformacije, no mogu se provoditi za najbitnije u

pogledu sigurnosti i kritičnosti za kontinuirani rad sustava. Rano testiranje štedi novac, jer mane koje se pojavljuju u kasnijem životnom ciklusu sustava su skuplji nego u ranim fazama tijekom izrade. Grupiranje klastera ukazuje da većina mana proizlazi od malog djela rješenja, primjerice vanjske biblioteke. Paradoks pesticida kao što pesticidi koji se više put koriste manje pomažu protiv nametnika jer nametnici unapređuju otpornost na otrov. Isti testovi ne mogu uvijek garantirati ispravan rad komponente jer se sustav stalno mijenja. Testiranje ovisi o kontekstu jer svaki sustav zahtijeva drugačije testove, tako će sustavi za web trgovina i društvena mreža imati drugačije testove. Posljednjo naćelo je zabluda o odsutnosti pogrešaka dokazuje da uklanjanje mana nije dovoljno. Svi zahtjevi moraju biti izrađen i ispravno raditi nije dovoljno da neki zahtjevi ispravno rade.

Bitno je spomenuti kako pristup koji se koristi za izradu rješenja utječe na testiranje. Primjerice za pristupe plansko razvijanje i inkrementalno razvijanje razlikuje se način provođenja testiranja. Kod planskog razvijanja česta je detaljna izrada zahtjeva iz kojih se mogu razviti testovi. Dok za inkrementalno razvijanje testovi se izrađuju za po jedan zahtjev. Jedna od prednosti planskog pristupa je da zbog detaljne specifikacije zahtjeva testovi mogu biti izrađeni cjelovito. Vremenski je zahtjevan planski pristup i time je prikladan za veće i kompleksnije projekte. Prednost inkrementalnog pristupa je razvijanje testova prije samog koda čime se postiže bolji kod. Inkrementalan pristup je namijenjen za manje do srednje projekte. Postoje dodatne podjele za spomenute pristupe. Ovisno o projektu razvojni tim će odabrati prikladni pristup koji najbolje odgovara projektu.

Važno je napomenuti kako testiranje ne može otkriti sve mane i previdjeti ponašanje u svim okolnostima. Testirati se mogu samo predloženi ulazi i usporediti s izrazima za specificirana ponašanja. Za ulaze i ponašanja koji nisu definiran mogu se pojaviti drugačiji rezultati i time mane. Čak i kada se uloži već broj testova i testera postoji mogućnost pojave mane. Testiranjem se smanjuju pojave mana. Klijenti ne moraju trošiti sredstva za financiranje mana. Poduzeće koje razvija programsko rješenje ne mora trošiti vrijeme na popravak mana i održava reputaciju poduzeća koje razvija pouzdana programska rješenja.

4.1. Funkcionalno

Funkcionalni zahtjevi direktno utječu na funkcionalno testiranje, kroz funkcionalna testiranja provjerava se dali programski kod zadovoljava funkcijske zahtjeve. Funkcionalno testiranje je testiranje što će sve aplikacija raditi u budućnosti tj. njezine funkcije. Funkcionalna i ne funkcionalna razlikuju se po namjeni testiranja. Funkcionalna testiranja testiraju rezultat, nefunkcionalna testiraju proces kojim je ostvareni rezultat. Primjerice funkcija aplikacije može biti unos nove školske godine za aplikaciju upravljanja studentima fakulteta. Važno je da je

funkcionalno testiranje zaduženo za provjeru funkcionalnih zahtjeva, ali nije zaduženo za ne funkcionalne zahtjeve. Provođenjem testiranja osigurava da rješenje zadovoljava klijentove zahtjeve. Funkcionalna testiranja iz pogleda detaljnosti mogu se podijeliti na: jedinična, integracijska, testiranje sustava i prihvatljivosti koja će biti razrađena u sljedećim poglavljima.

Prema [19] testiranje temeljen na specifikaciji i funkcionalno testiranje su sinonimu jer se specifikacija testova temelji na poznatom okruženju tj. poznatim podacima. Može se reći da se provodi testiranje u okruženju s definiranim granicama i očekivanim ponašanjima. Testiranje je neovisno o implementaciji i testiranje se može provoditi tijekom same izrade programskog rješenja. Naravno postoje i nedostaci poput izrade istih testova i rupe u testiranju zbog izrade testova koji pokrivaju samo specificirane scenarije. Funkcionalna testiranja osiguravaju kvalitetno rješenje jer rješenje radi ispravno i pouzdano. Prije konačnog izdanja pogreške su identificiraju i riješe. Testiranja su prilagođena domeni i time poštovani standardi i pravila. Zbog toga funkcionalna testiranja su krucijalna kod razvijanja kvalitetnog programskog rješenja jer zbog njih rješenje radi kao očekivano što u konačnici doprinosi klijentima.

4.1.1.Jedinično

Jedinično testiranje (eng. unit testing) u kontekstu objektno orijentiranog programiranja je proces kojim se provjeravaju jedinice: metode, klase i objekti. [3] Jedinično testiranje predstavlja testiranje najosnovnijih gradivnih elementa programa tj. jedinica. Upravo su metode, klase i objekti gradivni elementi svakog objektno orijentiranog programskog rješenja bilo za mobilne, desktop ili web rješenja. U radu s objektima moraju se testirati sve metode, vrijednost objekata i testirati promjene stanja. Metode koje se testiraju ne mogu biti uvijek testirane kao metode klase. Naslijeđene metode unose kompliciraniju dinamiku jer naslijeđena metoda može ovisiti o atributima objekta koji nasljeđuje. Za metode koje ovise o implementaciji prethodnika mora se uključiti cijelo stablo ovisnosti u testiranje. Cilj testiranja je potvrditi da svaka jedinica radi što se očekuje.

Prije izrade testa planira se koji kod će biti testiran. Za planirane testovi se izrađuju testne funkcije. Testovi se izvrše i provjere dobiveni izlazi i riješe mane. Rasprostranjena je praksa koristiti dvije vrste testova za provedbu testiranja. Prvi koji testira općeniti rad aplikacije i osigura da jedinica radi kao predviđeno. Dugi test koji je zadužen za neočekivana ponašanja i unose. Karakteristike koje ukazuju na ostvarenja kvalitetnih testova su sljedeći: izolirano testiranje s pripremljenim podacima bez ovisnosti, testiranje koda koji utječe na rješenje, a ne svake linije koda, revizija za svaki individualni test i često testiranje u ranoj fazi izrade programskog rješenja. Testovi koji posjeduju karakteristike kvalitetnih testova pomažu u ranom otkrivanju pogrešaka čime se sprečava nastanak složenijih grešaka. Zbog ranog otkrivanja

troškovi popravka su smanjeni. Nadograđivanje i promjene nad rješenjem je redovitije jer testiranje osigura da novosti ne narušavaju postojeće funkcije.

4.1.2.Integracijsko

Integracijsko testiranje je proces testiranja više od jedne komponente. Skup klasa, metoda i objekata tvori jednu cjelinu tj. komponentu time je jediničnog testiranja korak iznad integracijskog testiranja u pogledu detaljnosti, a ispod testiranja sustava. Cilj je utvrditi kako se komponente ponašaju u interakciji jedna s drugom. [3] Pomaže u otkrivanju podataka koji su krivog formata ili programske logike. Mogu se pojaviti novi zahtjevi za razvijati i testiranje pomaže u osiguravanju da nova funkcija radi ispravno s postojećim i ne narušava ih. Vanjske komponente zadužene za primjerice pružanje podatke kroz testiranje su provjereni. Sustavi se sastoje od povezanih komponenata i integracijsko povezivanje osigurava da komponente rade harmonično i stabilno. Povezivanjem komponenti dobiva se bolja slika kako sustav radi. Ovisno o planu testiranja se provode na manjem ili većem broju komponenata. Testiraju se sučelja i otkrivaju situacije u kojima sustav manifestira manu.

Slično jediničnom testiranju integracijsko testiranje prolazi kroz faze: planiranja, izrada, izvršenje i recenzija dobivenih rezultata. Razlikuju se dva pristupa veliki prasak (eng. Big Bang) i inkrementalni koji se dalje može podijeliti na pristupe: dolje prema gore, gore prema gore i sendvič. Veliki prasak je proces u kojemu se testiraju sve komponente odjednom. Veliki prasak namijenjeni je za mane sustave, ne zahtijeva planiranje i brzo izveden. Iz druge strane nije namijenjen za velike sustave jer faza testiranja čeka da sve komponente budu razvijene i teško je odrediti gdje se manifestirala greška. Inkrementalno testiranje je proces testiranja logičko povezane grupe komponenata koji završi kad su sve komponente napravljene i testirane. Najveća prednost inkrementalnog pristupa naprema velikom prasku je lakše pronalaženje greške zbog manjeg broja povezanih komponenata. Nedostatak pristupa je da zahtijeva opširno planiranje za određivanje povezanih komponenti i određivanje koji skup komponenti prije razviti.

Podvrsta pristupa dolje prema gore prvo testira komponente niže razine i onda komponente više razine. Komponente niže razine su komponente zadužene za elementarne radnje, nisu kompleksne i zadužene su za jedan zadatak. Primjerice komponente za provjeru unosa su komponente niže razine. Iz druge strane komponente više razine su kompleksnije i šireg opsega primjerice komponenta obrade plaćanja. Taj pristup koristi se za sustave koji su osjetljivi na mane i većinom se sastoje od komponenata niže razine. Pristup gore prema dolje je suprotan pristupu dolje prema gore. Namijenjen je za sustave čija funkcionalnost leži u komponentama više razine i bitne su nam povratne informacija korisnika. Sendvič pristup je kombinacija dvaju pristupa. Prednosti su fleksibilnost i osiguravanje da komponente niske i

visoke razine budu dobro testirane. Zbog korištenja dvaju pristupa vrlo je bitno dobra organizacija i planiranje kod testiranja i komunikacije među članovima tima. Razvojni tim će odabrati pristup koji najbolje odgovara funkcijskim zahtjevima i sposobnostima razvojnog tima.

4.1.3. Testiranje sustava

Predzadnja vrsta testiranje prije zadnjeg testiranja prihvatljivosti je testiranje sustava, a to je proces koji provjeri ispunjenost funkcijskih zahtjeva na razini cijelog sustava. [3] Cilj je osigurati da su zahtjevi ispunjeni u cijelosti bez mana. Prije izdanja rješenja provodi se testiranje i smanjuju mane zbog kojih bi se upropastilo korisničko iskustvo. Cijene popravke iz perspektive novca i vremena je vrlo niska, jer rješenje nije još izdano. Ne funkcionalni zahtjevi poput performanse obrađuju se u sljedećim poglavljima. Kad je moguće testiranja su provedena od odvojenog tima testera koji nisu sudjelovali u izradi rješenja kao programeri ili dizajneri. Zbog odvojenosti tima testeri mogu nepristrano provoditi testiranja. Testni slučajevi se i dalje razvijaju za funkcionalne zahtjeve, sučelje prema drugim sustavima mora raditi bez greške. U slučaju manifestacije mane rješenje se ne smije ugasiti ili zblokirati već obavijestiti korisnika. Testni slučajevi prolaze kroz faze: odabir funkcionalnog zahtjeva, izrada, izvršenje i revizija. Prednosti provođenja testiranja sustava su brojni. Prvo je stvaranje pouzdanje za ulagače u sustav jer je razvojni tim isporučio sustav prema zahtjevima. Drugo za sustav je dokazano da uspješno radi s drugim sustavima. Jedan od nedostataka je vremenski ulog jer zahtijeva planiranje i organiziranje testera čime se stvaraju troškovi.

4.1.4. Testiranje prihvatljivosti

Posljednje testiranje za funkcionalne zahtjeve je testiranje prihvatljivosti i process je provjere spremnosti rješenja za krajnje korisnike. Ključno je uključivanje korisničke perspektive kao mjerilo uspješnosti rješenja. Testiranje je provođeno od osoba koji testiraju izrađene korisničke zahtjeve. Testiraju se funkcionalni zahtjevi i određuje stupanj uspješnosti izrađenog rješenja. Rezultat provođenja testa prihvatljivosti je informacija dali je dovoljno dobro rješenje za korištenje u operativni okolini. Informirani izvođači mog bolje donijeti odluku kako postupati s daljnjim razvojem. U slučaju pada testa prihvatljivosti ne ispunjeni zahtjevi ili krivo ispunjeni ispravljaju se i provode kroz testiranja. Uspješni rezultat testiranja znači za izvođače da mogu krenuti u daljnje faze razvojnog procesa poput isporuke i održavanje. Podaci se većinom razlikuju od testnih jer testni podaci rijetko kad mogu predočiti stvarne podatke koji se koriste u sustavu. Podaci i postupak kojim se unose i manipuliraju je drugačiji od očekivanih i planiranih.

Potencijalno najbitnije testiranje s perspektive klijenta je testiranje prihvatljivosti, jer se pokazuju njihovi zahtjevi kao rješenje. Klijenti će se osjećati bolje jer znaju u kojem stanju se

nalazi rješenje i koje zahtjeve treba drugačije realizirati, ukloni ili čak dodati. Klijenti imaju očekivanja i pretpostavke za gotov sustav, no kad se izradi i isproba tek onda se može dobiti realna slika. Zahtjevi koji su se smatrali bitnim mogu se ispostaviti da se mogu zanemariti. Također se mogu javiti zahtjevi koji nisu bili specificirani, a doprinijeli bi sustavu. Testiranje je provođeno od razvojnog tima no najbolje od odvojenog ima ako je moguće, jer se time izbjegnu pristranosti. Uz odvojeni tim utjecaj klijenta u proces prihvatljivosti je od velikog značaja. Kako su klijenti stručnjaci u domeni i krajnji korisnici oni će bolje simulirati stvarne podatke i procese korištenja. Stupanj uključenja ovisi o klijentu i time može biti veliki i mali.

4.2. Ne funkcionalno

Gledano prema klijentskim zahtjevima izvršila se podjela zahtjeva funkcionalne i ne funkcionalne. Naizgled povezani pojmovi razlikuju se po aspektu zahtjeva. Iz jedne strane funkcionalne zahtjevi usredotočeni su na sami program, dok se ne funkcionalni bave pitanjima kako program radi. Primjerice funkcionalni zahtjev registracija ima popratne ne funkcionalne zahtjeve poput sigurnosti. Za registraciju vrlo je važno da završetkom korisnik dobije sigurne pristupne podatke. Tijekom provođenja registracije želi se izbjeći kreiranje zaporke koja je slaba. Domena testiranja ne funkcionalnih zahtjeva vrlo je široka i bavi se provjerom raznim provjerama aspekata rješenja. U radu su obrađena testiranja: sigurnosti, performanse, korisničkom sučelja i kompatibilnosti.

Ovisno o domeni za koje se sustav razvija neka ne funkcionalna testiranja postanu važnija. Po prirodi neki će sustavi biti složeniji s razgrađenom dubinom i širinom, dok će drugi biti jednostavni bez velikog broja funkcija. Složeni i podatkovno osjetljivi sustavi morat će zadovoljavati veći stupanj sigurnosti od sustava za upravljanje knjigama. Za složene sustave testiranja moraju biti u što većem broju i detaljnija. Važno je spomenuti da ne funkcionalni zahtjevi budu definirani na kvantitativan način kako bi se moglo provoditi kvalitetno testiranje. U testiranja zahtjevi se moraju opravdati argumentima koji se baziraju na dokazima. Primjerice u sustavu za iniciranje inzulina zahtjev da maksimalna doza ne smije prelaziti preporučenu dozu. Testiranje može biti vođeno provjerom tipa: doza ne prelazi preporučenu količinu sto puta. Uspješno provođenje za svako ponavljanje osigurava ispunjenost ne funkcionalnog zahtjeva. Prilikom izrade testova za sustave u domeni zdravstvo i vojske kriteriji primjerice sigurnosti bit će definirani stručnjaka u domenama.

4.2.1. Testiranje sigurnosti

Sigurnost kao jedna od najbitnijih ne funkcionalnih zahtjeva, za koju klijenti obraćaju veliku pažnju i zabrinutost. Rješenje će utjecati na razinu sigurnosti koja je potrebno razviti i

testirati. Testiranje sigurnosti može se definirati kao proces kojim se osigurava da rješenje zadovoljava nefunkcionalne zahtjeve. Kao u uvodnom poglavlju za ne funkcionalno testiranje treba obratiti pažnju da specificirani zahtjevi utječu na testiranje. Ovisno o detaljnosti sigurnosnih zahtjeva će se razviti sigurnosna testiranja, što znači da će testovi pokrivati spomenute slučajeve ne sve moguće. Sigurnost se može postići samo do određene mjere i ne može biti potpuna, jer postoje razne ranjivosti poput hardverske, softverske i socijalne. Treba usmjeriti napor u stvaranju rješenja koje pruža funkcionalnosti kada je sigurnost ugrožena.

Napadači koji žele pristupiti osjetljivim podacima ili ugroziti sustav imat će više vremena u pronalasku ranjivosti nego su imali razvojni programeri za izgradnju. Naime sustavi se moraju razviti do određenog roka i time ograničavaju vrijeme i opširnost testova. Zbog ograničenog vremena, dobra praksa je razviti testove i osigurati sigurnost za kritične elemente sustava prvo i dalje razviti manje kritične elemente. Postoje pristupi kojima se provede testiranje sigurnosti. Statična analiza provjerava kod kojim je napravljen program i model za koji provjerava formalne izjave. Prolazi se kod i provjerava ako postoje neke od najčešćih grešaka za specifični programski jezik. Kod se također ručno provjerava, dok se nasumični podaci ubacuju u program i provjere rezultati i zabilježe anomalije. Kada je to moguće dobro je uključiti odvojeni tim koji igra ulogu napadača i pokuša otkriti i iskoristiti ranjivosti.

4.2.2.Testiranje performanse

Testiranje performanse je proces utvrđivanja razine ispunjenost ne funkcionalnih zahtjeva performanse. Cilj je osigurati normalan rad sustava tijekom napora. Testiranje se provodi nakon što je sustav izrađen jer se onda mogu testirati komponente tijekom rada. Potrebno je otkriti maksimalnu razinu napora pod kojim sustav normalno radi. Otkrivanje gornje razine provodi se povećanjem broja transakcija i podataka koji su u sustavu u jednom trenutku. Prvo se osigurava da sustav zadovoljava specificirane zahtjeve i nakon se simulira opterećenje. Drugim riječima stvara se stres i pokuša otkriti gornja granica stresa. Za sustav koji je namijenjen da obrađuje sto transakcija, prvo se simulira s manje od sto transakcija, a zatim s većim brojem od sto. Testiranje se provodi sve dok se ne pronađe razina u kojoj sustav ne uspijeva obraditi transakcije i normalno funkcionirati. Maksimalni stres se simulira više put kroz duže vrijeme što otkriva probleme poput curenja memorije. Podaci korišteni u testiranju moraju što bolje pokrivati stvarne podatke koji se koriste u sustavu.

Potrebno je proći podfaze testnog procesa: planiranja, pripremanja okruženja, izvođenja, analize i optimizacije. U planiranju se odrede predviđane granice i količina transakcija. U pripremi simulira se rješenje u radnoj okolini i pokušaju pripremiti stvarni podaci. Izvode se prije spomenuti testovi i zabilježe metrike poput korištenih resursa. Nakon izvođenja

analiziraju se metrike i izrađuju izvješća koje sumiraju promjene u radu sustava. Ovisno o rezultatima izvjštaja provode se unapređenja i rješavaju problemi ako je to potrebno.

4.2.3.Testiranje korisničkog sučelja

Kao za prijašnja testiranja kroz testiranje korisničkog sučelja provjerava se u kojoj mjeri je rješenje ispunilo klijentske ne funkcionalne zahtjeve. U praktičnom smislu testiranje korisničkog sučelja je utvrđivanje ponašanja elemenata s kojima se susreće korisnik. Elementi mogu bit: meniji, alatne trake, pretraživanje, gumovi, boje i slično. Usko povezani pojam korisničko iskustvo se razlikuje od korisničkog sučelja. Korisničko iskustvo je subjektivno i bavi se kako se korisnik osjeća tijekom korištenja. Prate se korisnička mišljenja, doživljaji i u kojoj mjeri su elementi doprinijeli povećanju ili smanjenju istih. Cilj korisničkog iskustva je stvaranje ugodnog i pouzdanog korisničkog iskustva. Iz druge strane testiranje korisničkog sučelja zaduženo je za obezbjeđenje da elementi rade za što su namijenjeni. Osiguravanjem ispravnog rada funkcionalnosti unaprijeđuje se korisničko iskustvo, i time direktno utječe sučelje na iskustvo. Testiraju se aspekti: funkcionalnosti, vizualnog dizajna, responzivnosti i provjera ispravnog tipa podataka.

Sustavi mogu realizirati komunikaciju s korisnicima preko sučelja na razne načine, koji od njih će biti odabrati ovisi o ciljnim korisnicima i namjeni sustava. Prema [20] sučelja se mogu podijeliti na sljedeće vrste: grafičko, naredbenog retka, glasovno korisničko sučelje i sučelje koje prepoznaje geste. Testiranja će se razlikovati za glasovna i sučelje za geste teže se provodi testiranje jer geste i glas se razlikuje od osobe do osobe i geste se mogu razlikovati od pretpostavljenih. Naravno bitno je spomenuti sučelja proširene realnosti i sučelja za komunikaciju između ljudskog mozga i računala. Ova sučelja su vrlo nova i zahtijevaju veliku pažnju i obzirnost prilikom izrade testova. Prema [3] testiranje se može prevesti ručno i automatizirano. Ručno se testiraju elementi: polja, gumbi, padajući izbornici, tabele, meniji, poruke obavijesti. Automatizirana testiranja se temelje na korisničkim scenarijama korištenja za koja su razvijene skripte. Ručna testiranja će biti temeljitija, ali zahtijevati dodatno vrijeme za prevođenje. Automatizirana testiranja su brža u testiranju velikih kompliciranih sustava, ali mogu preskočiti neki element.

4.2.4.Testiranje kompatibilnosti

Prema [17] testiranje funkcionalnosti je provjera u kojoj mjeri novi sustav može izvoditi funkcionalnosti starijeg sustava. Većina novih sustava se razvija koristeći dijelove pa čak i cijele sustave koji se prilagode domeni. Prema širokoj definiciji sustavi se mogu smatrati: operacijski sustavi i web preglednici. Sustavi se razvijaju u okruženju i to okruženje čine programi i računalno sklopovlje. Provođenjem testiranja kompatibilnosti dobivaju se rezultati

koji usmjeruju razvojni tim za koji okruženje se mora prilagoditi sustavu. Operacijski sustavi: Windows, Linux i Mac su jedan dio programske potpore. Gotovo svaka aplikacija rasprostranjena je na raznim operacijskim sustavima i uređajima i time se mora provesti testiranje kompatibilnosti. Hardverske Konfiguracije se razlikuju po memoriji i procesorima za koje se mora provesti testiranje. Za sustave se testira kako funkcioniraju s drugim programima poput antivirusnih te se testira rad s vatrozidom.

Testiranje kompatibilnosti u domeni weba od interesa su preglednici i uređaji. Cilj testiranja kompatibilnosti je osigurati da se sustav izvršava jednako na skupu preglednika i uređaja. Postoje razne verzije i modeli uređaji i ne može se osigurati jednaki rad sustava na svim postojećim uređajima i preglednicima. Zbog tog razloga odrede se uređaji, preglednici i njihove verzije za koje se želi osigurati kompatibilnost i time dobije skup za koji sustav mora jednako raditi. Dobra je praksa prevesti testiranje nad skupom za svaku novu verziju sustava. Osiguravanje kompatibilnosti unapređuje aspekte: korisničkog iskustva i zadovoljstva, smanjenje resursa i povećava pouzdanost u sustav. Testiranje se izvodi prije samog lansiranja gotove verzije sustava, započinje u dizajn fazi. Faze dizajna i razvoja čine interval u kojem testiranje kompatibilnosti može početi. Unutar intervala pronađene greške se brže, jednostavnije i jeftinije mogu riješiti. Popravci otkrivenih grešaka u gotovoj verziji sustava su sporiji, kompleksniji i skuplji nego u ranim fazama razvojnog ciklusa. Općeniti koraci testiranja kompleksnosti mogu biti: određivanje skupa, planiranje, izvršenje testova, popravci i izvještaj. Planiranje obuhvaća određivanje važnih karakteristika poput brzine i željene rezultate. Testiranje se provodi ručno ili automatski i rezultati se evidentiraju u izvješću.

4.3. Ručno testiranje

Prije spomenuta funkcionalna i ne funkcionalne testiranja mogu se u određenoj mjeri provoditi ručnim testiranjem. Prema [3] ručno testiranje može se definirati kao proces izvođenje predefiniranih podataka i uspoređivanje s očekivanim rezultatima. Ručno testiranje provodi se od testera nad sustavom tijekom životnog ciklusa programskog rješenja. Kroz faze ručno se provodi svaka aktivnost od pisanja testnih scenarija, testova i testnih izvještaja. Programi za automatizirano testiranje se ne upotrebljava. Testovi zaduženi za pokrivanje gotovo svih funkcionalnosti su izrađeni ručno. Ručno testiranje može smatrati kao jedna od osnovnih aktivnosti programskog razvitka za otkrivanje grešaka. Svrha korištenja ručnog testiranja otkrivanje greški koje mogu biti zanemarene automatiziranim testiranjem.

Ručno testiranje se može podijeliti na sljedeće testiranje bijele, crne i sive kutije. Prema [17] bijela označava mjera u kojoj testni scenariji mogu pokrivati programski kod. U pristupu bijele kutije nije nužno testirati prema specifikaciji. Programer prolazi kod liniju po liniju i

utvrđuje njegovu ispravnost. Provjeravaju se izjave, putevi i kontrolne strukture. Za izjave i varijable se utvrđuje u kojoj mjeri su definirane u skladu s pravilima programskog jezika. Putevi označava kako se program može sve kretati s obzirom na izjave i kontrolne strukture. Programsko rješenje sastoji se od velikog broja kontrolnih struktura i izjava testiranje svakog puta je nemoguće postojećom tehnologijom.

Testiranje crne kutije provodi testni tim nakon što je programer proveo testiranje bijele kutije, no prepoznate greške rješava programer. Prema [17] testiranje crne kutije je proces uspoređivanje korisničkih zahtjeva s izrađenim funkcionalnostima. Testeri zanemaruju način kojim je izrađena funkcionalnost, njihov cilj je pronaći situacije u kojima se program ponaša drugačije nego što specifikacija diktira. Testeri su u većini slučaja i programeri koji su izradili funkciju zbog ograničenja vremena i resursa, najbolja praksa je odvojiti razvojni i testni tim. Tehnikom promjena stanja promatra se ponašanje funkcije s raznolikim ulazima popu funkciju prijave. Tehnika povjera granica osigurava da program proizvodi potrebne podatke i ponašanje.

Prema [21] testiranje sive kutije predstavlja kombinaciju bijele i crne kutije. Procesi dviju kategorija se spoje kako bi se dobila cijela slika i kombinacija prednosti. Tester u testiranju sive kutije zna osnovne dijelove i funkcije koje čine sustav no nije upoznat s cijelom implementacijom. Tester objektivno testira sustav i sve dijelove s djelomičnim znanjem implementacije. Jedna od tehnika je matrično testiranje koja utvrđuje koje varijable poprimaju vrijednost u programu i koje su neinicijalizirane i nekorištene.

4.4. Automatizirana testiranja

Potpuno automatizirana testiranja prema [22] su testiranja koja ne zahtijevaju ručnu intervenciju. Ručna intervencija su sljedeće kategorije aktivnosti: postaviti potrebnu okolinu, verifikacija i događaji. Okolina se mora postaviti preko API-a umjesto ručnog postavljanja čime se smanjuju moguće greške. Time su automatizirana testiranja suprotnost ručnim koji zahtijevaju ručnu intervenciju tijekom izrade i izvršavanja. Sami test mora imati ugrađenu logiku provjere ako su rezultati ispravni primjerice *Assert* metodom. Tijekom rada sustava događaju se raznoliki događaji poput: nestanka internetske veze, odabir gumba i nepovezanost s bazom podataka. Događaji se moraju programski izraditi koji simuliraju ručne događaje. Potpuna automatizirana testiranja su vrlo korisna u dobivanju povratnih informacija. Naime kada automatizirana testiranja zahtijevaju ručnu intervenciju povećavaju se potrebni resursi time i troškovi. Takvi testovi se neće ponoviti puno puta zbog asociranog troška.

Funkcionalna testiranja provode se za utvrđivanje funkcionalnih zahteva. Kao primjer jediničnog testiranja može se uzeti *AAA* (eng. Arrange, Act Assert) pristup. [3] Najprije se

prikupljaju podaci koji su koriste u testiranju i očekivani rezultati. Drugi korak testira jedinicu primjerice metodu i dobiva se rezultat. U posljednjem koraku uspoređuju se očekivani podaci s prvog koraka s dobivenim podacima dugog koraka. Posljednji korak daje do znanja programeru ako postoje mane u kodu ili sve radi ispravno. Prilikom prikupljanja podataka zbog aspekta brzine testiranja mogu se koristiti mock podaci. Za testove koji zahtijevaju podatke iz baze podataka bolje je koristiti mock podatke. Mock podaci su podaci koji su slični podacima iz baze koji bi se koristili u testu većinom se spremaju u polje ili listu.

Prema [23] integracijsko testiranje ili testiranje proizvoda je vrsta testiranja koje provjerava interakciju između komponentama. Za sustave koje nemaju veze prema drugim vanjskim sustavima tj. ne primaju ili šalju podatke integracijsko testiranje neće biti potrebno. Dijelovi poput baze i API-ji su dotične točke komponenata i za njih se može provesti testiranje. Mjera i način na koji će integracija biti provođena je definirana integracijskim planom. Općenito je bolje izraditi sustav koji ovisi o što manje u vanjskim komponentama, jer se time izbjegnu promjene zbog kojih dolazi do konflikta. U životnom ciklusu programskog rješenja integracijsko testiranje počinje rano u fazi izrade i traje do faze održavanja ako unosi kritične promjene. Time integracija ne prestaje i osigura da nove komponente ne narušavaju postojeće već unaprjeđuju sustavu. Automatizirano testiranje provedeno na cijelom sustavu zahtijeva vrijeme i iskustvo. Testira se od točke korisnika do programske i hardverske konfiguracije. Zbog ograničenih resursa provodi se testiranje za kritične funkcije poput obrade plaćanja. Integracijsko testiranje daje najbolje rezultate kad ga provodi iskusni testni tim.

Nefunkcionalna testiranja prema [23] se nazivaju i strukturalno testiranje provode se za ne funkcionalne zahtjeve koji definiraju koliko dobro sustav izvršava funkcije. Mogu se izraditi automatizirana testiranja za ne funkcionalne zahtjeve: sigurnosti, performansa, sučelja i kompatibilnosti. Testiraju se aspekti sigurnosti koji su: ranjivosti, analiza koda, skeniranje prijetnji. Ranjivosti mogu se pojaviti u pristupnim podacima, aktivnosti prije i nakon starta korisničke sesije. Prema [2] slabosti koje se mogu iskoristiti su: prijave, upravljanje ovlastima i administrativne ovlasti. Prema [24] testiranje performanse može biti vođeno testovima opterećenja sustava, gdje se automatski simulira povećani promet i podatke. Rezultati daju do znanja ako sustav zadovoljava zahtjeve, kako funkcionira pod stresom i koja je točka u kojoj sustav ne funkcionira ispravno. Testiranje sučelja simulira korisničko korištenje elemenata: unosa, gumba i menija navigacije. Testiranje kompatibilnosti automatizira se vlastitim ili postojećim okvirima i rješenjima. Matrica kompatibilnosti za operacijske sustave, preglednike i njihove verzije se mogu automatizirati izvršavanjem testnih scenarija i zapisivanje rezultate.

4.5. Usporedba vrste testiranja

Ručno testiranje je početno testiranje i obavezno za osiguravanje sustava bez brojnih greška. Ručna testiranja zahtijevaju poznavanje programiranja, razumijevanje zahtjeva te provođenje testiranja. Ručna će testiranja naprema automatiziranim biti sporija, ponavljajuća i skuplja. Prema [19] automatizirana su bolja opcija kod regresijskog testiranja kod kojeg se prošli testovi ponovno provode. Za testiranja koja se ponavljaju više puta ručna testiranja će biti manje pouzdana od automatiziranih. Naime test će biti provođeni koracima: unošenje ulaznih podataka, pokretanje testa i uspoređivanje rezultat s očekivanim. Zbog ponavljajućih aktivnosti ručni proces može biti dosadan za tester i rezultirati greškama.

Automatizirana testiranja zahtjeva razumijevanje okvira i programiranja bez toga se ne mogu izraditi testovi koji ispravno provjeravaju sustav i gubi se pouzdanost u rezultate. Smanjena pouzdanost je zbog ljudskog faktora, koji je sklon pogreškama. Prema [18] ručna testiranja neće biti moguće provoditi kod velikog broja regresijski testova. Veliki broj testova zbog količine može trajati dugo iako automatizirano. Slični testovi se mogu izbrisati ako testiraju isto u srhu bržeg izvršenja testova. Brisanje testova koji ne uzrokuju na greške nije preporučeno. Nove funkcije ili promjena postojećih može izazvati ponašanje koje je prije bilo pokriveno testom, a sad nije. Slijed koraka pokretanja ručnog testa specificiran je u testnoj skripti i mora se provoditi za svaki test ručno. S druge strane kod automatiziranih testova skripta je zadana od testera. Korištenje automatiziranog pristupa omogućuje primjenu istih koraka na više testova bez ručne intervencije i prilagodba za specifični slučaj.

Ručna i automatizirana testiranja su bitne za osiguravanje kvalitetno programsko rješenje. Za testiranje performanse i sigurnosti automatizirana testiranje će biti bolja, jer se moraju postići veći broj testiranja. Korisničko iskustvo će bolje biti testirano ručno zbog subjektivnog mišljenja testera. Korištenjem obiju vrsta testiranja omogućuje pokrivanje nedostataka koje imaju oboje. Primjerice testiranje prihvatljivosti može koristiti obje vrste testiranja, za provjeru funkcionalnih i nefunkcionalnih zahtjeva. Za situacije u kojima automatizirana testiranja ne pokrivaju i ne mogu simulirati korisničko ponašanje koriste se ručna testiranja. Jednako tako sustavi koji su komplicirani i zahtijevaju slična testiranja više puta koriste se automatizirana testiranja. Bilo koja vrsta testiranja se korist ne mogu se potpuno smanjiti i izbjeći pojava greške jer se mogu testirati samo specificirani zahtjevi.

4.6. Automatizirano testiranje i Selenium

U sklopu definiranja vrsta testiranja, njihove definicije, prednosti i nedostaci potrebno je spomenuti u kojoj vrsti testiranja pripada Selenium. Selenium svojim karakteristikama pripada u automatizirana testiranja. Definicija, prednosti, nedostaci i karakteristike alata Selenium će biti obrađene u nadolazećim poglavlju. Automatizirana testiranja su ručna testiranja izrađena programski koja se mogu ponoviti više puta bez potrebe ponovnog stvaranja testa. Jednom izrađeni testovi u Selenium-u iako zahtijevaju više vremena za izradu moći će se izvršiti više puta. Selenium može testirati ponašanje razvijenih testova u različitim preglednicima. Primjerice može se razviti testni scenarij za provjeru obrasca prijave, čiji rezultati se mogu koristiti za izradu izvješća. Automatizirati se može ljudsko ponašanje i time provoditi testiranje korisničkog sučelja. Jednom napisani testovi se ponovno pokreću za nove komponente ili starije koje su izmijenjene tj. Selenium podržava regresijsko testiranje. Zbog mogućnosti pisanja testova koji se mogu više put pokrenuti jedno pored drugog i regresijskog testiranja Selenium pripada u alate za automatizirana testiranja. Također Selenium podržava karakteristike automatiziranog testiranja to su: podržavanje stresnog testa i skalabilnost za veća i složenija rješenja.

4.7. Usporedba alata

Prilikom odabira alata prvi kriterij može biti dali je alat otvorenog ili zatvorenog koda. Otvoreni kod je razvijen od volontera koji dodaju nove funkcije i rješavaju greške, dok zatvoreni kod razvija poduzeće i prodaje licence. Licenciranje može biti kontinuiranim plaćanjem i jednokratnim plaćanjem. Naime prilikom sklapanja ugovora između razvojnog tima i klijenta, treba odlučiti dali će netko drugi preuzeti održavanje ili će tim nastavljati održanje. No postoji mogućnost da klijent želi gotovo rješenje bez daljnjeg održavanja. U slučaju da klijent želi provoditi održavanje s minimalnim troškovima razvojni ti će preferati otvoreni kod. Ako cijena nije bitan faktor ili ako se želi gotovo rješenje bez troškova razvojni tim odabire tip koda prema prednostima i nedostacima.

Najznačajniji kriteriji temelju kojih razvojni tim odabire alat za automatizirano testiranje će biti: podržani jezici, podrška, performanse, skalabilnost, jednostavnost korištenja i kompatibilnost. Razvoj web aplikacija može biti proveden raznim programskim jezicima od PHP-a do C#-pa. Podrška većeg broja jezika čini alat fleksibilnim za korištenje u raznim projektnim pothvatima. Podrška se realizira kao dokumentacija i korisnička podrška. Razvijena detaljna dokumentacija i dobra podrška stvara pouzdanost u alat. Stupanj formalne dokumentacije i podrške općenito će biti razvijeniji u zatvorenom kodu. Otvoreni kod će ovisiti

o razini uključenosti volontera, i aktivnosti zajednica koja doprinosi kvaliteti podrške. Performanse brzine i korištenih resursa bitne su u projektima s kratkim vremenskim rokovima. Veća brzina izvršavanje testova s manjem korištenjem resursa je poželjna. Alati koji podržavaju veliki broj kompliciranih testova su bolji jer se mogu koristiti u manjim i većim sustavima. Funkcije alata moraju ispravno raditi i biti prikazane pregledno. Pozitivno korisničko iskustvo bolje će rangirati alat. Alati kojima se mogu izraditi testovi u raznim okruženjima su poželjniji. Okruženja čine programi, hardver, i operacijski sustav. Odabrana su dva konkurenta za koje će biti razjašnjeni kriteriji, za Selenium izvojeno je posebno poglavlje za opis kriterija, karakteristika i što ga čini boljim odabir za ovaj projekt.

Prvi odabrani konkurent koji pruža potporu automatiziranog testiranja je Puppeteer. Prema [25] Puppeteer je višenamjenski alat kojim se može provoditi testiranje, izrada sadržaja i web scraping. Puppeteer izrađeni je od Googlea 2017 godine pretraživači Google i Firefox su podržani. Neke od karakteristika su izrada: jediničnih, integracijski testovi te testovi sučelja i performansa, izrada izvješća u HTML, XHTML ili JSON formatu. Alat podržava sljedeće radnje: automatizirani unos u polja, testiranje u pretraživaču, analiza pokrivenosti koda. Analizom pokrivenosti koda dobije se broj koji prikazuje zbroj svih izvršenih linija koda. Testovi se ne moraju odvijati jedan za drugim nego paralelno korištenjem Puppeteer Clustera što čini Puppeteer prilagodljivim većim brojem testiranja. Podržani je isključivo JavaScript ostali jezici nisu. Podrška je razvijena zbog velikog angažmana korisnika. Testiranja se provode bez grafičkog sučelja čime se postižu bolje brzine jer se grafički elementi ne moraju prikazivati. Puppeteer se temelji na DevTools Protocolu zbog čega olakšava krivulju učenja za osobe koje koriste DevTools. Puppeteer svojim API-evima puža veliku kompatibilnost pretraživaču Chrome i Chromium. Službeno ne postoji podrška za ostale pretraživače poput Safari i Edge.

Posljednji konkurent koji je razvijen za automatizirana testiranja je Cypress. Prema [26] Cypress je skup JavaScript alata za provođenje testiranja. Iako otvorenog koda razvijen je sjedeći pravila i norme. Mogu se izdvojiti poduzeća koje koriste Cypress, a to su: Johnson & Johnson, Autodesk i AirTable. Omogućuje izvršavanje testova direktno u preglednik i stvaranje izvještaja koji prikazuju nastale greške. Testovi su uključeni i dio aplikacije jer se izvode u samom web pretraživaču čime se povećaju performanse. Cypress može manipulirati DOM (eng. Document Object Model) objekti tijekom izvršavanja aplikacije. Kao i Puppeteer ne podržava druge jezike osim JavaScripta-a. Dokumentacija je razvijena od dobro povezane i aktivne zajednice. Testovi se ne mogu izvršavati paralelno u više od jednog preglednika koristeći Cypress Dashboard. Nadzorna ploča pomaže testeru u otkrivanju i otlkanjanje pogrešaka. Cypress podržava veći broj preglednika nego Puppeteer poput: Google Chrome, Firefox, Edge i Electron. Oba alata su prihvatljive opcije za web preglednike no ne mogu se koristiti za provjeru aplikacije koje su razvijene za računala ili pametne telefone.

Prednosti i nedostaci alata se mogu vidjeti u Tablica 1, kriteriji se navode u prvom lijevom stupcu, dok su konkurenti u prvom gornje redu. Kriteriji poprimaju kvalitativne vrijednosti za svaki od konkurenata. Kriteriji i pripadne vrijednosti su sažetak gornjeg teksta. Kada se kriteriji ocjenjuju pobjednik je Selenium. Razvojnom timu i pojedincima može biti Selenium od većeg interesa jer podržava veći broj programskih jezika i web preglednika uz održavanje istih kriterija kao konkurencija. Podržavanje više od jednog jezika omogućuje primjenu Selenium-a na veći broj raznoliki web aplikacija od jednostavnih web stranica do složenih rješenja. Troškovi i tip koda je za svakog konkurenta jednak i ne ističe se bolji. Prednost Selenium-a se može smatrati mogućnost izrade različitih izvještaja koji se lakše uklope u postojeću dokumentaciju i evidenciju.

Koji alat će biti odabran od razvojnog tima ovisit će o projektu kojeg razvijaju. Moraju se razmatrati zahtjevi, vremenski rokovi i postojeće znanje tima. Primjerice za sustav koji će biti implementirani u webu i neće biti prošireno na mobilne i desktop uređaje Selenium će biti najbolje od spomenutih rješenja za automatizirana testiranja. S obzirom na sve spomenuto najbolji alat za razvijanje ovog projekta je Selenium jer bitno nadmašuje konkurente prema kriterijima. No ne treba isključiti ostale konkurente poput Puppeteer-a i Cypress-a koji se u specifičnim scenarijima mogu bolje iskoristiti nego Selenium. Općenito Selenium će biti prvi i najpošteniji alat za automatizirana testiranja.

Kriterij/Alat	Puppeteer	Cypress	Selenium
Troškovi	Besplatno	Besplatno	Besplatno
Podržani Jezik	JavaScript	JavaScript	Java, C#, Python, Ruby, JavaScript
Vrsta Tetiranja	Jedinični, integracijski, sučelje, performanse	Jedinični, integracijski, sučelje, performanse	Jedinični, integracijski, sučelje, performanse
Izvještaji	HTML, XHTML, JSON	Izvještaji o greškama u pregledniku	Različiti formati izvještaja
Brzina	Testiranje bez grafičkog sučelja poboljšava brzinu	Izvršavanje testova unutar preglednika poboljšava performanse	Varira ovisno o konfiguraciji i pregledniku
Paralelno testiranje	Da, korištenjem Puppeteer Clustera	Ne, testovi se ne mogu paralelno izvršavati	Da, uz pomoć Selenium Grid
Krivulja učenja	Olakšana zbog DevTools Protocola	Može biti složenija, ali dobro dokumentirana	Varira ovisno o jeziku i kompleksnosti testa
Zajednica i dokumenacija	Varira, ovisno o angažmanu korisnika	Dobro povezana i aktivna zajednica	Opsežna i aktivna zajednica
Tip koda	Otvoreni kod (iako razvijen od Googlea)	Otvoreni kod	Otvoreni kod
Podržani Preglednici	Google Chrome, Firefox	Google Chrome, Firefox, Edge, Electron	Chrome, Firefox, Edge, Safari, Internet Explorer

Tablica 1: Konkurenti i Selenium, (Izvor: Vlastita izrada, 2024)

5. Selenium

Selenium je jedan od najkorištenijih alata za automatizirano testiranje web aplikacija na raznolikim web preglednicima. Selenium je razvijen 2004 godine od Jason Huggins kao unapređenje početne verzije, dobiva ime kao protuotrov protiv tadašnjeg testnog alata Mercury's QuickTest Professional. Prema [27] Selenium je zaživio kao pomoćni program koji je omogućio automatizirano unošenje podataka za jednu web stranicu. Naime Jason je stranici morao u sklopu svog posla popunjavati polja koja su predstavljala vremenske intervale. Prilikom unošenja novog reda intervala korisnik je morao čekati za serversku obradu što je smanjilo produktivnost. Zbog tog problema Jason Huggins je razvio pomoćni program koji je automatizirano unosio podatke iz čega se razvio Selenium.

Selenium u svojoj je ranoj verziji bio jednostavniji od konkurenata i postao popularan i kroz godine se postavio na tržište kao najčešće korišten alat za automatizaciju. Selenium je imao greške koje su utjecale na izvršavanje testova no zbog uključivanja korisnika u razvoj alata smanjile su se i ispravile greške te proširile funkcionalnosti. Skup novih funkcionalnosti se razvija od Simon Stewart tijekom 2007 godine pod nazivom Selenium WebDriver. Sljedeće godine su od interesa: 2007 razvoj Selenium RC, 2016 Selenium WebDriver3 i 2021 Selenium 4. Dobra kompatibilnost može se vidjeti po tome što Selenium WebDriver 2011 godine postane standardiziran od organizacije W3C. Naime W3C (eng. World Wide Web Consortiuma) je međunarodna organizacija koja propisuje standarde za web tehnologiju.

Prije spomenuti Selenium RC postaje zastarjeli i nadograđeni je WebDriverom verzijom Selenium 3.0. Automatizirani testni alat Selenium sastoji se prema [27] od troje alata: Selenium WebDriver, Selenium IDE i Selenium Grid. Selenium Web Driver je zadužen za komunikaciju s preglednikom. Web Driver se može smatrati kao posrednikom između testera i web preglednika. Korisnik koristi razumljiv jezik za izradu programa kojeg Web Driver prijevodi web pregledniku. Selenium IDE je alat implementiran kao Google i Firefox dodatak. Dodatak omogućuje korisniku snimiti interakciju sa stranicom i reproducirati. Snimaju se interakcije poput klikovi na gumbe i unošenje tipkovnicom. Snimke su automatski pretvorene u testne skripte koje se mogu eksportirati u Python i Javu. Selenium Grid je zadužen za izvršavanje testova istovremeno na više web preglednika. Stvore se čvorovi koji predstavljaju računalo s specifičnom konfiguracijom. Mogu se izvršiti testovi izrađeni u: C#, Python, Ruby i Javi za preglednike: Edge, Google Chrome, Safari i Mozilla Firefox.

Arhitektura Selenium-a 3.0 JSON Wire Protocol koji omogućuje prijenos podataka između klijenta i servera. Selenium 4.0 zamjenjuje JSON Wire Protocol, BiDirectional protokolom. Jednom kad se klijent i server povežu novi upiti ne zahtijevaju novu vezu već se

upiti odvijaju na uspostavljenoj vezi. Uvođenjem BiDirectional protokola ukloni se potreba za novom vezom i omogući instantna komunikacija i razmjena podatka.

Selenium ima broje funkcionalnosti i karakteristike jedna od njih je mogućnost manipulacije Shadow DOM komponentama. Elementi, njihov CSS stil i JS skripte mogu se grupirati u „shadow root“ time se stvore komponente odvojene od glavnog DOM-a. Izradom Selenium WebDriver-a 4.0 može se koristiti metoda *getShadowRoot* u kombinaciji s ostalim metodama: *SearchContextt*, *shadow_host* i *findElement* i za dohvaćanje svih i promjena elementa koji činie komponentu. Komponente izrađene Shadow DOM-om vrlo su interesantne programerima jer mogu fokusirati napore za izradu funkcije i stila za jednu komponentu činjena od više elemenata. Zbog toga što se komponenta ne nalazi u DOM-u stilovi i funkcije se ne prenose na ostale elemente i time ne mijenjaju ili ugroze ostale elemente.

5.1. Značajke

Izrađena anketa za Selenium [28] s 410 sudionika prikazuje zanimljive rezultate. Približno šezdeset posto sudionika ne preferira testiranje bez vizualnih elemenata. Rezultat dodatno čini Selenium boljim od ostalih jer podržava grafičko testiranje dok Puppeter ne podržava. Otvoreni kod čini Selenium vrlo interesantnom za razvojni tim jer nema troškove vezane uz licencu. Naime Selenium je odličan alat iz aspekta testiranje web aplikacije u preglednicima, te dodacima pruža potporu za testiranja u drugim okruženjima. Veći broj ispitanih sudionika testira u svim podržanim preglednicima, čak njih sedamdeset i osam. Broj nedostataka je vrlo maleni, nedostaci koji se pojave su većinom vezani uz dinamiku automatiziranih testiranja. Web preglednici se nadograđuju zbog čega se moraju ažurirati testovi. Testovi vezani uz sučelje morat će biti ažurirani u slučaju izmjene ili dodaje elemenata. Razvijeni Selenium testovi neće biti dovoljno u ostvarenju pouzdanog rješenja bit će potrebno izgraditi ručna testiranja. Selenium se istaknuo od svih spomenutih konkurenta kao najbolji alat za provođenje automatiziranih testova.

Prije spomenuti Selenium IDE pripada u kategoriju alata za snimanje i reprodukciju. Snimke su vrlo korisne početnicima za stvaranje automatiziranih testova bez potrebe znanja programskih jezika. Selenium se može primijeniti u više načina nego samo testiranja. Selenium se može koristiti za sakupljivanje podataka iz web stranica. Može se izraditi sakupljač koji će pretražiti stranice u okolini i prikupljati broj kuća koje se prodaju. Na temelju podatka može se izradi aplikacija koja prati stanje kuća, koja bi bila korisna osobama u okolici za brži pronalazak kuće preko filtriranje i pretraživanja. Umjesto ručnog pregledavanja kada se snizi cijena artikla može se izraditi skripta koja daje obavijest kad se snizi cijena artikla.

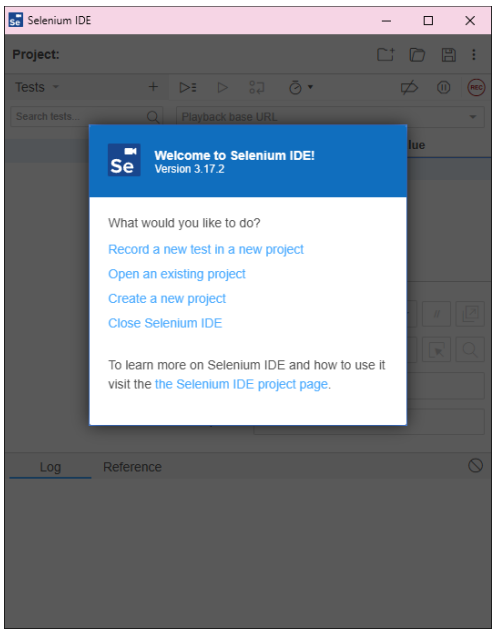
Selenium WebDriver rješava problem koji se javlja kad DOM element nije učitan, a test zahtjeva taj element. [27] Javlja se greška, razlog greške mogu biti raznoliki preopterećen server, neoptimizirani kod ili brzina izvršavanja testa. Kao rješenje problemu mogu se koristiti strategije čekanja: implicitna, eksplicitna i prilagodljiva. Strategije čekanja osiguraju da element bude učitan i „spreman“ za korištenje. Implicitna strategija pričeka određeno vrijeme prije nego što šalje iznimku za nepostojeći element. Čekanje je definirano globalno i može rezultirati samo uspjehom ili neuspjehom. Eksplicitna strategija uvjetima definira kada je element spreman za korištenje. Primjerice element tip gumb se može provjeriti stanje ako se može kliknuti na njega. Za svaki element se mogu izvršiti provjere i osigurati da je učitan. Prilagodljiva strategija nadograđuje eksplicitnu strategiju time da nudi veću kontrolu upravljanja procesom provjere elementa. Primjerice umjesto da se element provjeri jednom može se provjeriti više puta dok se ne ostvari uvjet.

5.2. Opis korištenja

Prošlo poglavlje Značajke 5.1 spominje između ostalog karakteristike Selenium IDE-a. Zaključno poglavlje o Selenium-u bavi se instalacijom i korištenjem alata Selenium IDE. Za testiranje web aplikacije koristi se Selenium IDE i zbog toga obradit će se koraci instalacije i objasniti način korištenja. Prije sve potrebno je preuzeti dodatak Selenium IDE. Dodatak se može preuzeti za Google Chrome preko Chrome Web Store-a. Ostali pretraživači mogu preuzeti Selenium IDE preko njihovih trgovina poput Firefox-a pomoću Firefox Browser Add-Ons. Da bi se instalacija alata mogla provesti potrebno je zadovoljiti uvjete. Prvi uvjet je imati instalirani operacijski sustav Windows, Linux ili macOS. Drugi uvjet instalirani pretraživač Google ili Firefox. Ovi uvjeti čine Selenium IDE vrlo neovisnim alatom jer ne ovisi o velikom broju alata i vanjskih resursa. Naime neovisnost alata o drugim alatima čini ga stabilnim i dostupnim. Alati koji imaju veliki broj ovisnosti od vanjskih resursa teže će reagirati na promjene vanjskih alata nego što će neovisniji alati moći.

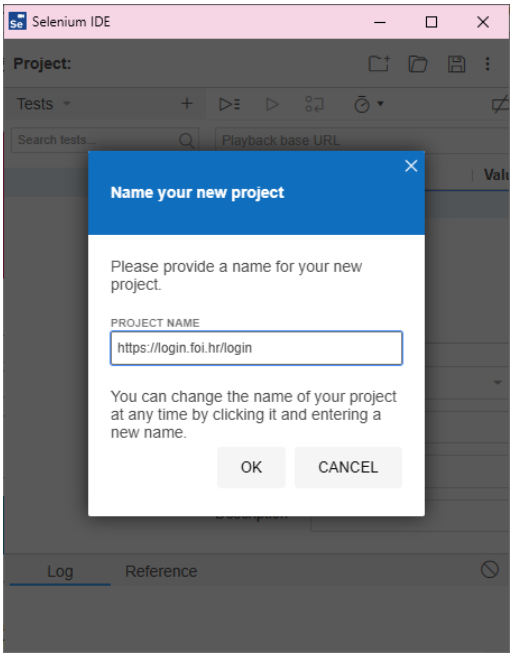
Preuzeti dodatak automatski se instalira. Dodatak se može prikvačiti u alatnu traku preglednika za brži pristup. Otvaranjem dodatka Selenium IDE iz liste dodatka može početi rad s alatom. *Slika 1* prikazuje početni prozor instaliranog Selenium IDE. Iz ovog prozora korisnik može vršiti brojne opcije prva opcija je za brzo testiranje „Record a new test in a new project. Druga opcija „Open an existing project“ omogućuje otvaranje postojećih projekata koja je vrlo korisna za već izrađene projekte sa svojim testovima. Treća opcija „Create a new project“ namijenjena je za grupaciju testiranja i spremanje u jedan projekt. Čim se postiže bolja organizacija testova i verzioniranje. Odabirom četvrte opcije izlazi se iz alata Selenium IDE.

Za početak i upoznavanje može se odabrati prva opcija dok za daljnje korištenje nad projektima odabir treće opcije će biti bolji.



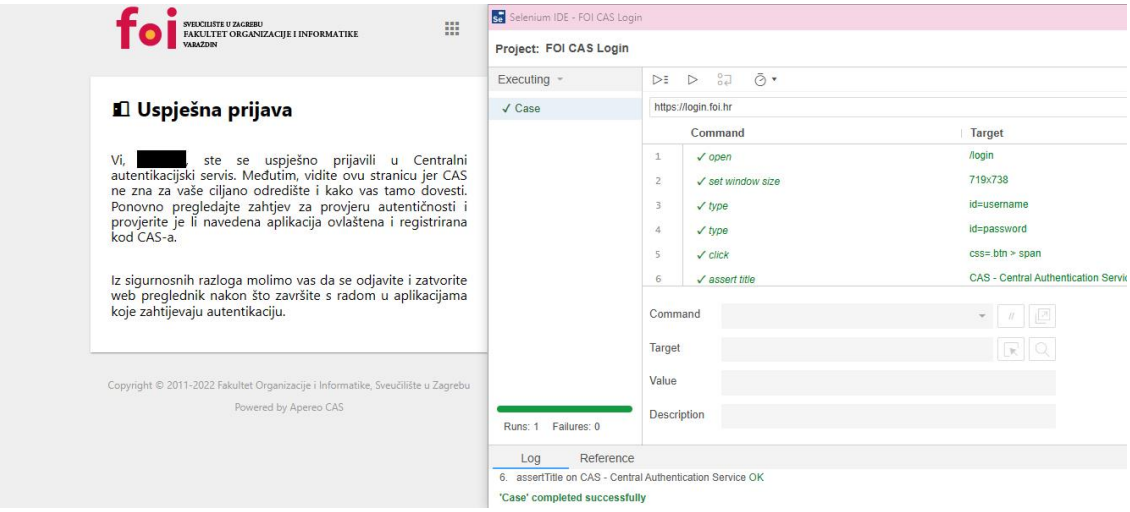
Slika 1: Početni prozor Selenium IDE-a, (Izvor: Snimka zaslona Selenium IDE, 2024)

U svrhu opisa korištenje izgrađeni je testni scenarij prijave u sustav ispravnim podacima. Nakon odabira „Record a new test in a new project“ upiše se URL web stranice koja se želi testirati u našem slučaju FOI CAS prijavu. Odabirom gumba „Start recording“ počinje snimanje. Snimati će se prijava u slučaju unosa korisničkog imena i zaporke koja postoji. Nakon unosa pristupnih podataka desnim klikom u prozoru odabire se „Selenium IDE“ te „Assert“ i nakraju title. Nakon snimanje se zaustavi klikom gumba „Stop recording“.



Slika 2: Izrada testa, (Izvor: Snimka zaslona Selenium IDE, 2024)

Jednom kreirani test može se pokrenuti u svrhe testiranja. *Slika 3* prikazuje prozor s rezultatom testiranja i alat Selenium IDE komande, cilj i vrijednost. Komande su u interakciji s aplikacijom neke od njih su: otvaranje, pisanje, postavi, odaberi i klikanje. Prema [27] postoji preko 90 komandi. Komande predstavljaju akcije koje korisnik može činiti tijekom interakcije s aplikacijom. Cilj predstavlja element nad kojim se vrši komanda. Ti elementi mogu biti vidljivi na stranici a da korisnik može biti u interakciji s elementom. Elementi poput padajućih izbornika, polja, gumbova i sličnih elemenata. Postoje različiti tipovi elementa neki poput linkova vode korisnika u novi prozor, element slike prikazuju sliku, tekstualno poja pokazuju tekst tako i numerička broj. Da bi Selenium IDE „prepoznao“ element koriste se identifikatori. Element može poprimati CSS *id* identifikator. Value predstavlja vrijednost koju poprima ciljni element, vrijednost u *Slika 3* je skrivena zbog sigurnosnih razloga. Testni scenarij „Case“ je uspješno izvršen što se može vidjeti po poruci „Case completed successfully“ također u prozoru je postignuti cilj prijave. Test se može lokalno spremiti u svrhe ponovnog pokretanja i testiranja funkcije prijave. Naravno testovi pohranjeni mogu se dalje spremiti i organizirati za bilježenje i dokumentiranje tijekom programskog razvoja. Prozor log pokazuje logove koji su vezani uz komande vrijeme kada je bila komanda izvršena i potencijalne greške uočene tijekom izvođenja. U desnoj strani prozora Selenium IDE-a nalazi se testni scenarij „Case“. Svi novi testni scenariji dodat će se desnoj strani. Nad gotovim testom mogu se vršiti operacije brisanja i promjena naziva. Odabirom opcije „+“ dodaje se novi test za koji se ponove prijašnji koraci. Postoje pogledi koji se mogu izmijeniti odabirom opcija iznad testova u obliku padajućih izbornika. Opcije su „Tests“, „Test Suite“ i treća opcija „Executing“. Prva opcija je polazna i prikazuje sve testove. Druga opcija služi da prikupljana slične testove i grupira ih prema jednom slučaju. Primjerice odvojeni testovi: prijava korisnika, dodavanje proizvoda u košaricu, plaćanje i odjava mogu se grupirati u slučaj korištenja korisnička kupovina.



Slika 3: Ponovljeni test, (Izvor: Snimka zaslona, 2024)

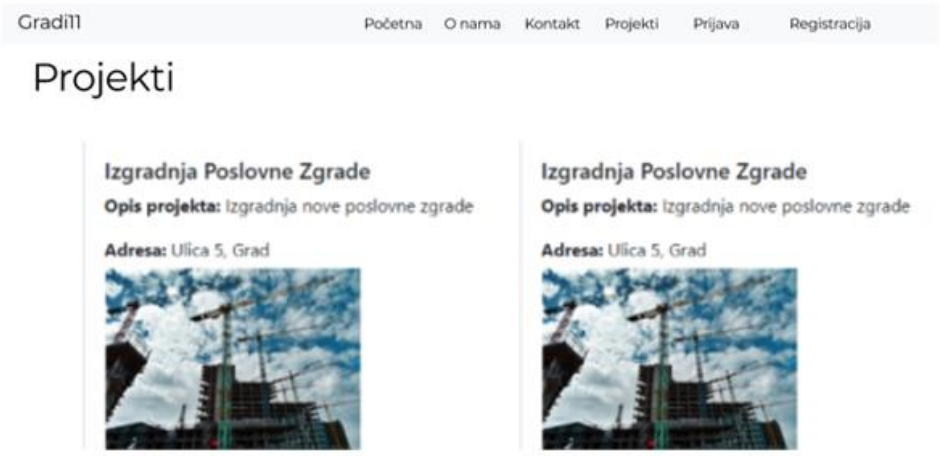
6. Praktičan rad

Praktičan dio rada bavi se testiranjem web aplikacije razvijene u programskom jeziku PHP uz korištenje CSS i HTML jezika. Koriste se okviri Bootstrap za jednostavno stiliziranje i obogaćivanje izgleda, te PHPMailer za slanje mailova prilikom registracije. Kao razvojna okolina odabrani je XAMPP sustav koji pruža funkcije lokalnog servera i sustava upravljanja bazom podataka. Testiranja će biti provođena dodatkom Selenium IDE. Testirat će se razni aspekti web aplikacije kako bi se osiguralo funkcioniranje i normalan rad korisnika s web aplikacijom. Testovi koji potvrđuju funkcionalnost će biti smatrani uspješnim ako se funkcionalnost izvodi ispravno od početka do završetka radnje. Testirat će se razni aspekti od prijave do stvaranje zapisa. Prema tome neće biti sveobuhvatni test za cijelu web aplikaciju nego pojedinačni testovi koji testiraju dio web aplikacije. Čime se osigura da pojedine funkcije ispravno rade i da su napravljene prema definiranim zahtjevima. Testiranja korisničkog sučelja su također izrađena koji osiguraju da korisnici nesmetano mogu pristupiti funkcionalnostima web aplikacije.

6.1. Opis aplikacije Gradi11

Web aplikacija Gradi11 predstavlja rješenje interesantno za poduzeća koja se bave izgradnjom građevinskih objekata poput: kuća, zgrada i pomoćnih objekata. Gradi11 namijenjena je registriranim korisnicima i neregistriranim koji autentifikacijom postanu ovlašteni pristupati aplikaciji i funkcionalnostima. Od interesa je Gradi11 jer nudi opcije izrade projekta i ocjenjivanje istih od prijavljenih klijenata u obliku recenzija. Poduzeće može vršiti temeljne operacije nad entitetima baze podataka kroz CRUD(eng. Create Read Update Delete) operacije preko formi gdje to ima smisla. Korišteni su entiteti koji bi mogli biti zanimljivi građevinskoj firmi poput: projekti, zahtjevi i recenzije. Cilj izrade web aplikacije Grad11 je pokazati kako se Selenium može koristiti u testiranju jedne web aplikacije i njezinih funkcionalnosti. Testiranje nije samo dokaz ispravnosti funkcija već i postupak kojim se uoče nedostaci i moguća unapređenja.

Skica aplikacije služio kao gruba reprezentacija zahtijeva u vizualnom obliku. Početna verzija kako bi aplikacija mogla izgledati prikazana je u *Slika 4*. U skici nisu definirane sve funkcije i skup podataka, već su korištene najbitnije funkcije i privremeni podaci. U skici prikazane su funkcije autentifikacije provođene izbornicima registracije i prijave te funkcija prikaza projekta. Skica je prva inicijalna ideja kako bi web aplikacija Gradi11 mogla izgledati s izrađenim funkcionalnostima. Izrađene su interaktivne skice za goste, klijente i administratore unutar kojih su poveznice koje vode do drugih prozora i funkcija.



Slika 4: Gradi11 web aplikacija, (Izvor: Snimka zaslona Figma, 2024)

6.1.1.Funkcijski zahtjevi

Kao prije spomenuto korisnički zahtjevi su vrlo važni u izradi web aplikacija, jer definiraju koje sve funkcije moraju biti izrađene. Korisnici imaju neke funkcionalnosti zajedničke dok su neke izrađene za specifičnu ulogu korisnika. Unutar aplikacije funkcionalnosti će biti dostupne korisniku ovisno o ulozi koja mu je dodijeljena. Uloge koje se mogu razlikovati su gosti, klijenti i administratori. Za uloge klijent i administrator zajedničke funkcije su: prijava u aplikaciju, izmjena profila i odjava iz aplikacije. Gosti se mogu registrirati i postati klijenti dok su administratori predefinirani. Također uloga gost posjeduje ograničeni broj funkcionalnosti a to su pregled projekata i recenzije. Izrada zahtjeva, profila i recenzije gostu nisu dostupne sve dok se ne registrira uspješno

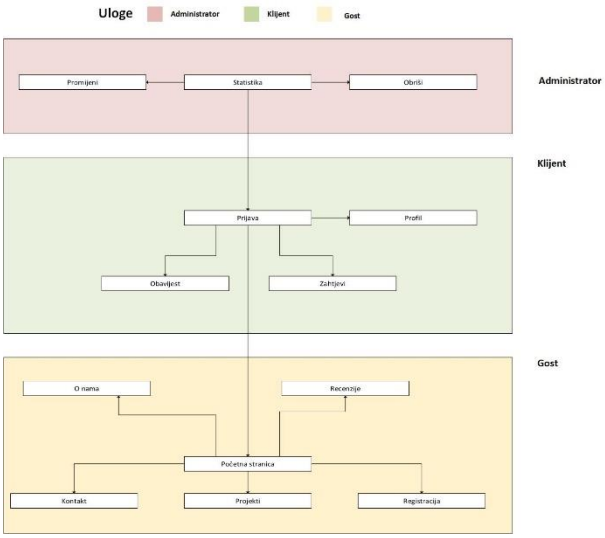
Klijent posjeduje veći broj funkcionalnosti od gosta. Razlika između administratora i klijenta su funkcionalnosti kojima imaju pristup, a ne samom broju dodijeljenih funkcionalnosti. Administrator će imati pristup funkciji izmjene projekta za koju klijent nema pristup, iz druge strane klijent može stvoriti recenziju a administrator ju jedini može obrisati. Klijent može stvoriti za dodijeljeni i završeni projektu samo jednu recenziju, jedan klijent ne može stvoriti više recenzija za dodijeljeni projekt. Jedna recenzija ima naziv, opis i ocjenu od jedan do pet. Klijenti mogu podnijeti zahtjeve za izradu projekta, pregledati projekte, stvoriti recenzije za projekte koji su završeni. Zahtjevi mogu imati naziv i opis koje zadaje klijent također slanjem zahtjeva automatski zahtjev poprima status na čekanju. Projekt se sastoji od zadatka koje korisnik može pregledati ako ga administrator postavi zadatak javni. Korisnik ne može mijenjati ili brisati zadatke. Zadaci koje je administrator postavio kao privatne korisniku se neće prikazati, i gosti ih također neće vidjeti. Projekti imaju naziv, opis, adresu, sliku i fazu. Klijent prima obavijest kada je zahtjev prihvaćeni od administratora. Klijent može imati više odobrenih projekta za koje može stvoriti recenziju kad je projekt završen.

Administrator ima većinu funkcionalnosti zadužene za upravljanje projektom i zadacima projekta. Poslani zahtjevi od klijenta mogu biti prihvaćeni i odbačeni. Administrator može izbrisati neprimjerene recenzije poslane od klijenta. Projekti su automatski stvoreni temeljeno na podacima zahtjeva: naziv i opis, te adrese profila klijenta. Projekti mogu biti izmijenjeni od administratora, može se promijeniti bilo koji od atributa od naziva do faze. Uz izmjenu profila omogućeno je brisane projekta. Također vezano uz projekte postoji funkcionalnost sortiranja prema fazi. Faze projekta prema kojim administrator može sortirati i svrstati projekt su: planiranje, priprema gradilišta, izgradnja, završena inspekcija i završen projekt. Za projekt administrator može postaviti zadatke i dodatno upravljati njima. Zadaci se mogu stvoriti za jedan projekt, obrisati i izmijeniti atributi: naziv, opis, status zadatka, i vidljivost. Uz pomoć atributa vidljivosti administrator zadaje ako klijent može ili ne može vidjeti zadatak. U slučaju kad administrator ne zadaje vidljivost zadatka ona će se postaviti u nevidljivo. Funkcionalnost statistike je realizirana za isključivo administratore kao prikaz: broj projekta po fazama, broj zadataka po projektu, prosječna ocjena po završenom projektu, broj zadatka po statusu i broj zahtjeva po statusu.

6.1.2. Navigacijski dijagram

Izrađeni je navigacijski dijagram *Slika 5* koji prikazuje kojim web stranicama ima pojedina uloga pristup. Gledano od gore prema dolje gornje stranice imaju pristup donjim stranicama. Administratori imaju pristup klijent stranicama i gost stranicama, klijenti mogu pristupati svojim i gost stranicama, dok uloga gost ima pristup samo svojim stranicama. Neke stranice će ovisno o ulozi korisnika prikazati drugačije sučelje i funkcije. Primjerice web stranica recenzije će klijentima omogućiti stvaranje recenzije gosti će moći samo pregledati recenzije. Administratori ne mogu stvoriti recenzije, no mogu ih obrisati te će sučelje biti drugačije za uloge. Boje se koriste kao pozadina pravokutnika, gdje jedan pravokutnik grupira stranice dozvoljene pojedinoj ulozi. Web stranice u žutoj boji pripadaju ulozi gostim, zelenoj boji pripadaju stranice klijenta, ružičastoj boji pripadaju web stranice administratora.

Iz dijagrama može se vidjeti da stranice: početna, o nama, kontakt, registracija, recenzije, projekti pripadaju ulozi gost. Gosti pregledavaju podatke koje su stvorili klijent i administrator. Klijenti imaju pristup stranicama: profil, zahtjevi i obavijesti. Klijent kao i gost može pregledavati stranice, dodatno može stvoriti recenziju, profil i zahtjeve. Obavijesti pripadaju isključivo klijentu. Administratori imaju pristup stranicama: statistika, promijeni i obriši. Stranica o statistici prikazuje podatke poput broja odobrenih zahtjeva. Administrator posjeduje pristup stranici koja je zabranjena za vršenje ažuriranja vezano uz projekte i zadatke. Stranica obriši vrlo je slična stranici promijeni po tome što je centralna za brisanje gotovo svih entiteta poput projekta, zadatka i recenzije.

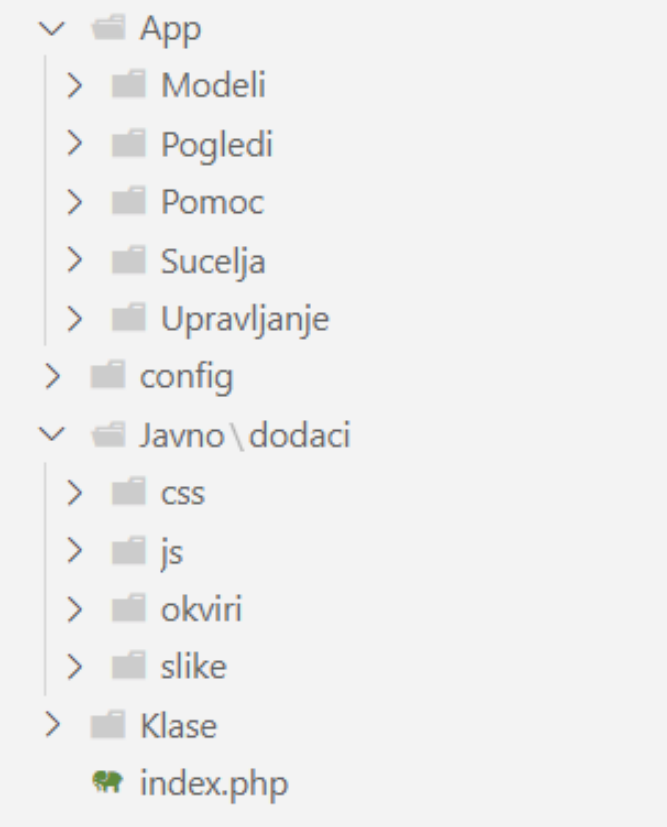


Slika 5: Navigacijski dijagram, (Izvor: Vlastita izrada MS Visio, 2024)

6.1.3.Pristup implementaciji

Pristup za izradu web aplikacije je objektni pristup točnije odabran je ORM(Object Relationship Model). Naime za svaki entitet je definirana entitetska klasa s metodama dohvaćanja i postavljanja vrijednosti, repozitorij koji definira metode obrade entitetnih svojstva. Umjesto proceduralnog pristupa odabrani je objektni pristup više razine apstrakcije koji omogućuje bolju održivost i nadograđivanje web aplikacije. Također se koristi pristup odvajanje modela podataka i prikaza. Model podataka čine entitetne klase i repozitoriji. Prikaz zaduženi je za prikaz podataka i sučelja za manipulaciju podataka. Manipulaciju nad podacima vrše se preko obrasca.

Zbog ovog pristupa izrađena je struktura direktorija kao u *Slika 6*. Model podataka implementiran je u mapi *Modeli* u kojoj se nalazi entitetne klase i repozitoriji. Mapa *Pogledi* sadrži dokumente koji uključuju ponavljajuće elementa prikaza poput zaglavlja i podnožja i pozivaju dokumente *Upravljanje* koji su zaduženi za poslovnu logiku. Neki od dokumenta iz mape *Upravljanje* će dodatno prikazati podatke elemente ovisno u namjeni. Za projekte pogledi će koristiti zaglavlje i podnožje te pozvati upravljanje namijenjeno za projekte. Upravljanje će za svaku od uloga biti drugačije prikazana primjerice administratori će moći promijeniti i obrisati projekte uz dodatak da mogu zadatke izraditi. Klijenti i gost mogu pregledati projekte. Iz slike može se vidjeti mapa *Javno* u kojoj se nalaze: slike, CSS, JS datoteke i okviri. Mapa config sadrži datoteke koje definiraju putanje za: klase, JS i CSS datoteke te datoteka koja definira metapodatke. Mapa *Klase* sadrži klase za povezivanje s bazom podataka, formatiranje rezultata iz PHP upita.



Slika 6: Struktura web aplikacije, (Izvor: Vlastita izrada MS Code, 2024)

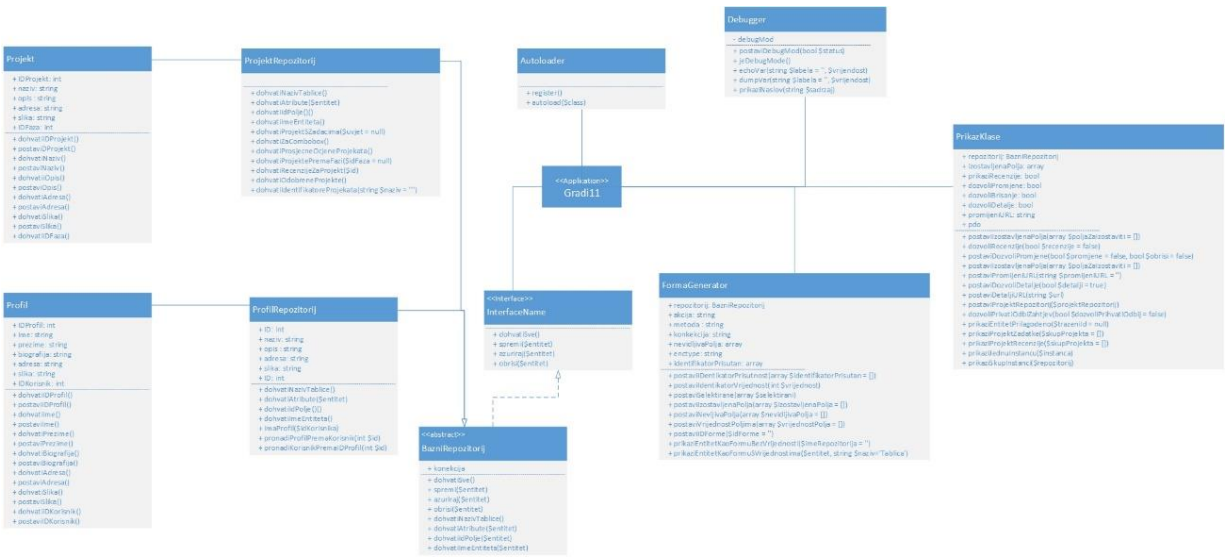
6.1.4.Dijagram klasa

U projektu izrađene su klase gdje je to moguće kako bi se smanjio ponavljajući kod i postiglo jednostavnije održavanje koda. Dijagram klasa *Slika 7* prikazuje klase i odnose između njima. Klase se mogu grupirati u nekoliko skupina: klase prikaza, klase modela, klase mapiranja i pomoćne klase. Klase prikaza poput *PrikazKlase* zadužene su za prikaz PHP rezultata koristeći HTML i CSS. Klase modela poput: Projekt, Recenzija, repozitoriji i druge modeliraju entiteta iz baze podataka kao objekte i metode kojima se vrše manipulacije nad podacima entiteta. Klase mapiranja su zadužene za uparivanja podatka iz jednog formata recimo iz obrasca s formatom klase jedna od klasa je *MapiranjeFormeNaEntitet*. Posljednje klase su pomoćne poput *PovezivanjeBazom* kojom se vrši povezivanje aplikacije s bazom podataka.

Dijagram na *Slika 7* prikazuje klase i odnos između njih, prikazane su najvažnije klase koje reprezentiraju projekt. Klase repozitorija su vrlo slične, pa nisu sve prikazane. Izrađeno je sučelje *RepozitorijSucelje* koje deklarira obavezne metode koje moraju implementirati klase koje ga koriste. Sučelje propisuje metode, no njihova implementacija izrađuje se u klasama koje ga koriste. Izrađena je apstraktna klasa *BazniRepozitorij* koja implementira većinu obveznih metoda sučelja *RepozitorijSucelje* i definira dodatne metode koje su od interesa za

određene grupe klasa. Klase koje proširuju apstraktnu klasu mogu prepisati metode apstraktne klase i dodati vlastite metode koje nisu definirane u apstraktnoj klasi. Klasa *ProjektRepozitorij* će koristiti nasljeđene metode *spremi*, čija je implementacija u apstraktnoj klasi. Konkretna klasa *ProjektRepozitorij* izrađuje vlastite metode. Jedna od metoda koja obrađuje dio statističke funkcionalnosti je metoda *dohvatiBrojZadatakaPoProjektima*. Metoda se odnosi na *ProjektRepozitorij* i nije dostupna drugim klasama. Zbog ovakve strukture i odnosa između klasa kôd se može ponovno iskoristiti, fleksibilan je u pogledu proširivanja klasama i specifičnih metoda.

Dijagram u *Slika 7* definira odnos između klasa. Klasa *BazniRepozitorij* implementira *RepozitorijSucelje* i one su u odnosu realizacije. Klase *ProjektRepozitorij* i *BazniRepozitorij* su u odnosu generalizacije gdje *ProjektRepozitorij* nasljeđuje metode *BazniRepozitorij*. Nadalje klasa *Projekt* i *ProjektRepozitorij* su povezane asocijacijom. Klase: *BazniRepozitorij*, *ProfilRepoztorij* i *Profil* također su povezne asocijacijom. Nadalje klase *Autoloader*, *Debugger*, *FormaGenerator* se koriste u web aplikaciji. Klasa *Autoloader* zadužena je za uključivanje namespace-a pozvane klase u radnu klasu, primjerice pozivanje klase *Profil* u *ProfilRepozitorij*. Klasa PrikazKlase prikazuje PHP rezultat u kombinaciji s HTML-om i CSS-om. Nadalje klasa *FormaGenerator* generira obrasce bez vrijednosti te one s unaprijed popunjenim podacima. Klasa Debugger ima pomoćnu ulogu za ispis vrijednosti varijable i jednostavni prikaz unutar web aplikacije.



Slika 7: Dijagram klasa, (Izvor: Vlastita izrada MS Visio, 2025)

6.1.5.Opis odabranih dijelova koda

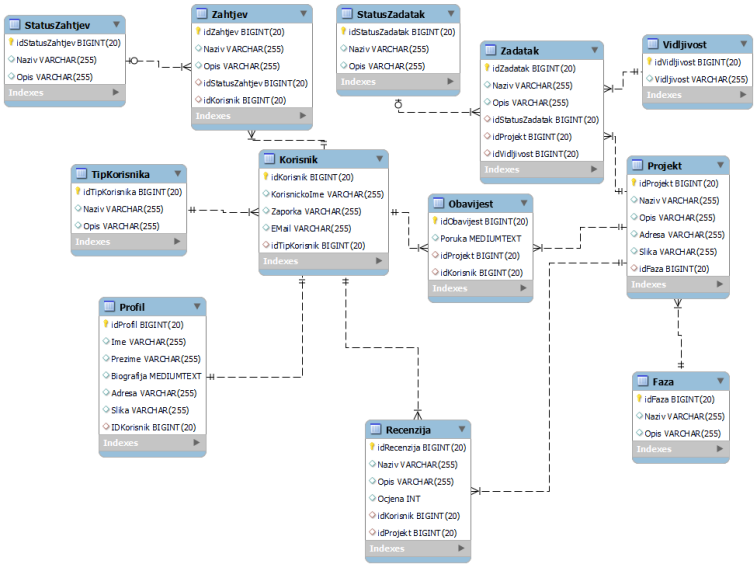
Jedan od zanimljivijih blokova kôda koji se može izdvojiti iz projekta je metoda *spremi*. Metoda pripada apstraktnoj klasi *BazniRepozitorij*, koja je odgovorna spremanju entiteta u bazu podataka. Parametar metode predstavlja objekt koji pripada jednom od definiranih tipova, odnosno entitetskim klasama. Povratni tip metode je *bool*, a rezultat se može provjeriti kako bi se utvrdilo je li metoda uspješno izvršila spremanje objekta u bazu podataka. Svaki repozitorij implementira obavezne metode definirane u apstraktnoj klasi koje opisuju entitet u bazi nazivom tablice i atributima. Varijabla *$upit* definira *SQL* unos u bazu podataka. U bloku *try catch* pokušava se izvršiti upit. Ako je uspješan vraća true, a ako nije, vraća false. Također se koriste svojstva klase za ispis stanja pojedinih dijelova koda. Primjerice svojstvo *$debug* klase *BazniRepozitorij* ispisuje *SQL* upit koji će biti izvršen. Svojstva se također koriste za bilježenje grešaka. Greška s kodom 23000 bilježi specifičnu poruka, dok se ostale greške bilježe općenitom porukom. Poruke o greškama i uspješnim operacijama mogu se prikazati korisniku pomoću HTML-a i CSS-a, pozivanjem odgovarajućih metoda. Kao što je prije napisano zbog naslijeđenih metoda klase repozitorija mogu koristiti metodu spremi za spremanje objekta u bazu podataka *$repozitorij->spremi($objekt).*

```
public function spremi($entitet): bool {
    $nazivTablice = $this->dohvatiNazivTablice();
    $atributi = $this->dohvatiAtribute($entitet);
    ...
    $upit = "INSERT INTO $nazivTablice (" . implode(", ",
             array_keys($atributi)) . ") VALUES (:"
            . implode(", :", array_keys($atributi)) . ")";
    $this->debugger->echoVar("SQL upit:", $upit);
    try {
        $stmt = $this->konekcija->prepare($upit);
        $stmt->execute($atributi);
    } catch (PDOException $e) {
        if ($e->getCode() === '23000') {
            $this->greskeC->postaviGreskuKorisnik("Kljuci", "Ne može
            se izraditi zapis s nepostojećim ovisnostima!");
        } else {
            $this->greskeC->postaviGreskuKorisnik("Baza podataka",
            "Greška: " . $e->getMessage());
        } return false;}
    ...
    return true;}
$repozitorij->spremi($objekt);
```

6.1.6.Baza podataka

Baza podataka je integralni dio svake web aplikacije. Prvo je izrađen ER *Slika 8* model koji je temelj i putokaz za izradu tablica u SQL jeziku. ER (Entity Relationship) model opisuje entitete i relacije između njih. Tijek izrade aplikacije započinje deklaracijom tipova koje atributi mogu poprimiti, dok su vanjski i primarni ključevi proizvoljni. Entiteti su pomoću SQL jezika izrađeni kao tablice, relacije kao ograničenja a atributi kao svojstva tablice. Nakon što su uneseni podaci, skripte entiteta i vrijednosti su uvožene u phpMyAdmin.

Gotovi svi entiteti povezni su vezom 1:M kao što je slučaj za entitete *Zahtjev* i *StatusZahtjev*. Status zahtjeva može pripadati i odnositi se na više zahtjeva, dok zahtjev može imati samo jedan status. Vrlo slično je odnos *Zadatak* i *StatusZadatak* gdje se status zadatka odnosi na više zadatka, a jedan zadatak može imati samo jedan status. Entiteti *Korisnik* i *TipKorisnika* su u odnosu 1:M tip korisnika tj. uloga se odnosi na više zahtjeva no jedan korisnik može imati tj. pripadati samo jednoj ulozi. Entiteti *Profil* i *Korisnik* su u odnosu 1:1 jedan profil pripada samo jednom korisniku i jedan korisnik može imati samo jedan profil. Entiteti *Recenzija* i *Korisnik* su u odnosu 1:M jedna korisnik može izradi i više recenzija, no jedna recenzija može biti izrađena od jednog korisnika. Entiteti *Vidljivost* i *Zadatak* su u odnosu 1:M vidljivost se odnosi za više zadataka, a jedan zadatak može imati samo jednu vidljivost. Entiteti *Projekt* i *Zadatak* su u odnosu 1:M gdje jedan projekt sadrži više zadataka, no jedan zadatak pripada jednom projektu. Entiteti *Projekt* i *Faza* su u odnosu 1:M jedan projekt pripada jednoj fazi i jedna faza može se odnositi na više projekta. Entiteti *Korisnik* i *Obavijest* su u odnosu 1:M gdje jedan korisnik može primati više obavijesti, dok jedna obavijest pripada jednom korisniku. Posljednji odnos entiteti *Projekt* i *Obavijest* su u odnosu 1:M jedan projekt odnosi se na više obavijest, ali će jedna obavijest biti vezana samo za jedan projekta.



Slika 8: ER baze podataka, (Izvor: Vlastita izrada MySQL Workbench.io, 2024)

6.1.7. Korišteni alati

Za realizaciju web aplikacije se koristi XAMPP i njegovi alati te VS Code. XAMPP je popularno razvojno okruženje za razvijanje lokalnih i serverskih aplikacija. XAMPP je odabran kako je operacijski sustav Windows. Između ostalog XAMPP i njegovi alati kroz nekoliko koraka instalacije pripreme razvojnu okolinu za izradu web aplikacije. Kako bi se skriptni jezik PHP mogao pokrenuti i izvršiti naredbe potrebno je pokrenuti skriptu na serveru. XAMPP može pokrenuti PHP skriptu, izvršiti instrukcije i rezultat prikazati na lokalnom serveru. Baza podataka također se izvršava na serveru što omogućuje interakciju klijenta, servera i baze podataka.

Pisanje se vrši alatom VS Code, koji nudi potporu za brojne programske jezike. Alat se može proširiti instalacijom dodataka za specifične jezike. Može se izdvojiti dodatak PHP IntelliSense, koji pomaže u pisanju koda kroz sugestiju ključnih riječi. Podržane su i vlastite klase, njihove metode i svojstva koja se nadopunjuju kod pisanja koda. Također je omogućena dokumentacija u obliku komentara koji opisuju klasne metode. VS Code može otvarati individualne datoteke, a projekti se mogu otvoriti kao jedna direktorijska struktura s više datoteka. Alat je odabran zbog gotovih funkcionalnosti poput pretraživanje cijelog projekta i prozora prikaza direktorija i datoteka. Nadalje, funkcionalnost dokumentiranja koda od velike je pomoći u održavanju i nadograđivanju postojećeg kôda.

6.2. Testiranje aplikacije Gradi11

Testiranje aplikacije je realizirano s pomoću Selenium IDE dodatka, točnije korišten je Google Chrome dodatak. Iako se testovi izvode automatizirano za njihovo kreiranje potreban je ručni rad. Da bi se testni scenarij automatizirao, koraci se moraju ručno izvesti i snimiti pomoći Selenium IDE dodatka. Primjerice, za stvaranje novog projekta moraju se unijet podaci, spremiti i osigurati da se unos stvarno izvršio i zapis dodao u bazu podataka. Podaci se najprije unose ručno preko obrasca. U nekim testovima podaci će se pripremiti pritiskom na gumb, kao u testu *Administrator briše projekt*. Automatizirana testiranja moraju biti ponovljiva, tj. moraju se moći izvršiti više puta. Da bi se test smatrao uspješnim svaka od komandi koja pripada testu mora biti izvršena. U grafičkom sučelju Selenium IDE-a boje označavaju je li test uspješan ili ne. Testovi koji su označeni crvenom bojom i imaju pokraj križić nisu uspješni. Greška se može pronaći u logu. Zelena boja i kvačica označuje testove koji su uspješni.

Da bi se test mogao izraditi, prvo se pripreme podaci. Sljedeći korak je koristiti podatke u testu, a zatim očistiti pripremljene podatke i rezultate. Svaki test je cjelina i testira specifičnu funkciju. Neki pripremljeni podaci se mogu koristiti u jednom testu, dok neki mogu biti važeći za više njih. Primjerice većina testova će koristiti pripremljene podatke za projekt, poput testova *Administrator briše projekt* i *Administrator izmjenjuje projekt*. Jedan test mora pripremiti sve potrebne podatke što uključuje i preduvjete. Test, poput *Korisnik izmjenjuje profil*, ima preduvjet prijave, jer izmjena profila zahtjeva prijavu korisnika u sustav. Nakon prikupljanja podataka, testiranje obrađuje prikupljene podatke i testira funkcije sustava. Krajnji korak gotovo svakog testa je čišćenje testnih podataka, kako ne bi došlo do konflikta s postojeći podacima sustava u daljnjem testiranju. Kada test pođe sve tri faze - priprema, obrada i čišćenje - može se ponoviti više puta i neće biti u sukobu s drugim testovima.

Kao što je spomenuto u teorijskom djelu, testiranja su podijeljena na funkcionalna i ne funkcionalna testiranja, a svaka od tih kategorija ima četiri podvrste. U praktičnom dijelu rada, za ne funkcionalna testiranja odabrano je testiranje korisničkog sučelja. Veća važnost posvećena je ovom testiranju jer Selenium IDE provodi testiranje na temelju vizualni elemenata. Testiranja korisničkog sučelja primarno su zadužena za interakciju korisnika s vizualnih elementima: provjerava se jesu li dostupni i preusmjeruju li poveznice ispravno. Funkcionalna testiranja bit će druga vrsta testiranja koje kombinira jedinično i integracijsko. Naime, testiraju se metode klasa, i to ne samo jedne klase, nego više klasa, zbog čega se podvrste mogu kombinirati. Funkcionalna testiranje bit će brže izvedeno i provjerit će radi li aplikacija kako treba. Dakako se isprepliću vrste testiranja, no u radu pokuša se izdvojiti vrste testiranja.

Testovi sučelja ovise o vizualnim elementima i strukturi korištenoj u web aplikaciji. Kako bi se smanjila ovisnost o vizualnim elementima i povećala ponovljivost testiranja uz testiranja korisničkog sučelja izrađeno je funkcionalno testiranje. Sveukupno je izrađeno tridesetak testova *Tablica 2* koji pokrivaju gotovo sve funkcionalnosti Gradi11. Svi testovi se mogu pokrenuti odjednom i ukupno traju dvadesetak minuta. Manji testovi poput Prijava Klijenta traju manje zbog manjeg broja komandi. Testovi poput *Administrator izmjenjuje zadatak* traju duže jer imaju veći broj komandi. Kako bi se smanjilo trajanje testova izrađena su funkcionalna testiranja kod kojih se priprema i čišćenja podataka provodi se u nekoliko pritiska na gumbove umjesto interakcije s vizualnim elementima web aplikacije. Svi testni slučajevi koji su korišteni u Selenium neće biti objašnjeni. Pojedinim testovima posvećena je veća pažnja zbog toga što ostali testovi rade vrlo slično. Nadalje neki testovi imat će komande koje koriste gotovo svi testovi kao test prijave. Testovi koji koriste specifične komande bit će posebno istaknuti. Na kraju testovi koji uključuju veliki broj komandi iz drugih testova također izdvojeni i detaljno opisani.

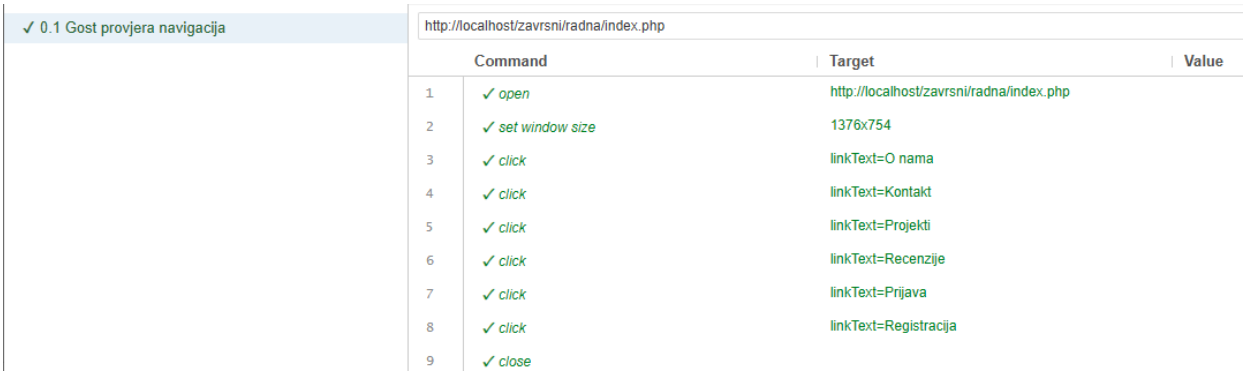
Testovi korisničkog sučelja	Funkcionalni testovi
Gost provjera navigacija	Klijent izmjenjuje profil
Prijava Klijent	Klijent stvara zahtjev
Korisnik izmjenjuje profil	Klijent izrađuje recenziju
Klijent neispravna prijava	Klijent pregledava dodijeljene projekte i zadatke
Klijent izrađuje recenziju	Klijent pregledava obavijesti za dodijeljene projekte
Korisnik šalje zahtjev	Klijent pregledava poslane zahtjeve, odobrene, odbačene i poslane
Administrator izmjenjuje profil	Administrator briše projekt
Prijava Administrator	Administrator sortira projekte
Administrator mijenja projekt	Administrator izmjenjuje projekt
Administrator obriše recenziju	Administrator odobrava zahtjev
Administrator odbije zahtjev	Administrator briše recenziju
Administrator prihvaća zahtjev	Administrator odbacuje zahtjev

Administrator stvara javne zadatke	Administrator dodaje projekte i zadatke
Administrator stvara privatne zadatke	Administrator izmjenjuje zadatak
Administrator izmjenjuje zadatak	

Tablica 2: Izrađeni testovi, (Izvor: Vlastita izrada, 2025)

6.2.1. Testni scenarij provjera navigacije

Prvi test *Slika 9* osigurava da gost može pristupati ovlaštenim stranicama s pomoću izbornika. Nadalje provjerava se preusmjeravaju li poveznice korisnika na ispravno odredište. Odredište komande *click* usmjeruje korisnika na stranice koje moraju biti dostupne gostu kao poveznice u izbornik. Kako je prije spomenuto jedan od funkcionalnih zahtjeva je pružanje gostu da može pregledati projekte i recenzije. Funkcijski zahtjev se realizirao u obliku web stranica gdje jedna prikazuje projekte a druga recenzije. Nadalje nije strog definirano no poželjno je da gost može pristupiti dijelu o nama i kontaktu. Štoviše korisnici koji nisu prijavljeni su gosti i moraju imati pristup web stranici prijave, te je testirani link prijave. Također za buduće klijente provođeno je testiranje linka registracije. U slučaju izostavljanje poveznice iz navigacije komanda za otvaranje poveznice neće se moći izvršiti i test će pasti. Test prema tome može obavijestiti gdje leži greška i mogu se poduzeti koraci koji uklanjaju grešku.

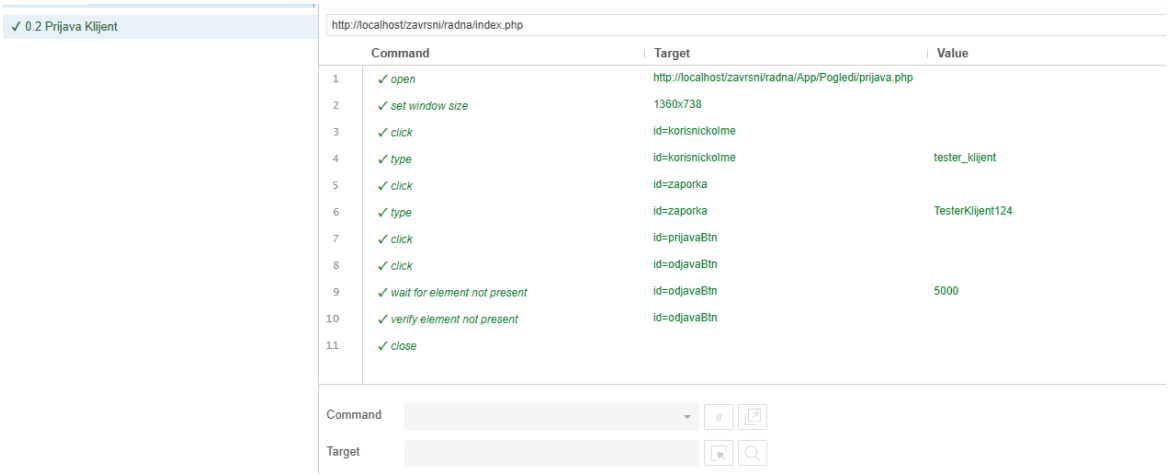


Slika 9: Testni scenariji provjera navigacije, (Izvor: Snimka zaslona, 2025)

6.2.2.Testni scenarij prijava

Prijava je vrlo važna funkcionalnost gotovo svake web aplikacije. Prijavom korisnik dobiva pristup web aplikaciji Gradi11. Prijava *Slika 10* je vrlo bitna jer gotovo svi testovi će koristiti komande prijave, jer većina funkcija su rezervirane autentificiranim korisnicima. Potrebno je unijeti ispravne autentifikacijske podatke: korisničko ime i zaporka. Prva komanda *open* otvara stranicu prijave gdje se korisnik prijavljuje. Sljedeća komanda *set window size* postavlja dimenziju prozora u pixelima s pomoću širine i visine. Cilj tj. *target* definira stvarnu vrijednost koja je za ovaj test 1360 pixela u širini i 738 pixela u visini. U testu prijave prva faza priprema podatka vrši se unosom korisničkom imena i zaporke. Stvarne vrijednost korisničkog imena i zaporke zadaju se u *value* dijelu komandi *type*. Kao cilj koristi se identifikator za zaporku je *korisnickoIme* a za zaporku *zaporka*. Funkcionalnost prijave se izvrši klikom na gumb. Posljednja faza čišćenja je odjava iz aplikacije pritiskom gumba odjavi.

Nakon odjave, komandama verify *wait for element not present* i *verify element not present* provjeri se je li se korisnik odjavio iz sustava. Naime komanda *wait* pričeka pet sekundi da se element ne učita. Komanda *verify* provjeri da element nije učitan u *DOM* objektu. Prva komanda pričeka pet sekundi jer se želi stvoriti pufer za neučitavanje elementa. Komanda *verify element not present* dodatno osigura da element ne postoji u web stranici prijava. Naime korisnik koji je odjavljeni nema funkcionalnost odjava. Odjavom i provjerom dali je odjava bila uspješna osigura se odjava iz sesije i uklanjanje konflikta s drugim testovima koji koriste sesije. Korisnik je u ulozi klijent s korisničkim imenom *tester_klijent* i on neće biti izbrisan. Korištenje testnog korisnika sa stalnim podacima smanjuje se ovisnost. Primjerice da se koriste korisnici stvarnog sustava. Klijent bi koji ima račun koji se koristi u svrhe testiranja bio mogao smetati u testiranju. Nadalje testovi koji koriste klijentov račun bi ugrozile korisničko iskustvo kada bi testovi izvršava radnje a korisnik je prijavljen.



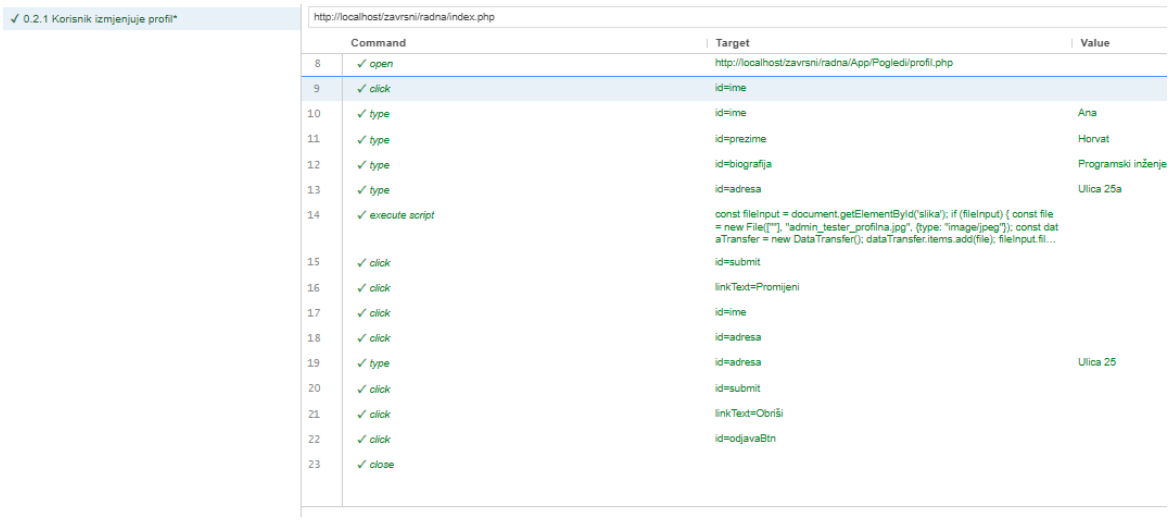
Slika 10: Testni scenariji prijava, (Izvor: Snimka zaslona, 2025)

6.2.3.Testni scenarij korisnik izmjenjuje profil

Test izmjenjuje profil je istaknuti jer uključuje komande koje se koriste u testu prijave i izrade profila. Zbog toga što su komande prijave objašnjene u testu prijave izostavljene su iz *Slika 11*. Komande *type* kao cilj uzimaju: ime, prezime, biografiju i adresu. Vrlo zanimljiva komanda je *execute script* koja kao cilj ima JS skriptu. Naime pomoću skripte u cilju izvrši će se uvoz slike umjesto ručnog uvoza stvarne slike. Skripta će stvoriti virtualnu datoteku i pridružiti je input elementu s ID-jem *slika*. U varijablu *fileInput* dohvaća i sprema se element s DOM ID-jem *slika*. Uvjet provjerava ako varijabla *fileInput* sadrži DOM element. U slučaju da je if uvjet istinit stvori se varijabla *file*. Varijabla *file* se definira konstruktorom *File*. Definirala se prazna datoteka koja ima naziv *admin_tester_profilna* i MIME tipa je *image/jpeg*. Skripta koristi JavaScript DataTransfer API za manipulaciju nad datotekama. Objekt dataTransfer je instanca klase DataTransfer. U liniji *fileInput.files=dataTransfer.files* varijabli *fileInput* dodaje se kao popis datoteka virtualnu popis datoteka. Događaj *changeEvent* zadužen je za događaj promjene datoteke. Argument *bubble: true* prenosi u roditelje elemente događaj, što znači da roditelji mogu osluškivati element. Na kraju *fileInput* izvrši *changeEvent* događaj kroz metodu *dispatchEvent()* što omogućuje da se izvrši *onchage* događaj za element.

```
const fileInput = document.getElementById('slika');
if (fileInput){
const file = new File([""], "admin_tester_profilna.jpg", {type:
"image/jpeg"});
const dataTransfer = new DataTransfer();
dataTransfer.items.add(file);
fileInput.files = dataTransfer.files;
const changeEvent = new Event('change', { bubbles: true });
fileInput.dispatchEvent(changeEvent);
}
```

Izmjenjuje profil je prvi test koji mora detaljnije razraditi fazu čišćenja. Naime nakon prijave koja je preduvjet ovog testa slijedi faza pripreme. Ovaj test kao i ostali testovi koji imaju preduvjet prijave koristit će komande testa prijave. U prvoj fazi pripreme, stvori se profil preko obrasca. Da bi se test smatrao uspješnim korisnik može stvoriti samo jedan profil i zbog toga razloga nakon izrade profila isti mora biti obrisan. Brisanje profila izvodi se pritiskom gumba koji se može zapaziti u komandi *click* i vrijednosti cilja *linkText=Obriši*. Klikom na gumb izvrši se skripta koja briše profil testnog korisnika. Posljednji korak je odjava korisnika. Takav test proći će kroz korake: prijave, stvaranje profila, brisanje profila i odjave iz aplikacije. Zbog tako strukturiranog testa on se može ponoviti više puta i neće biti ostatka u obliku testnih podataka.

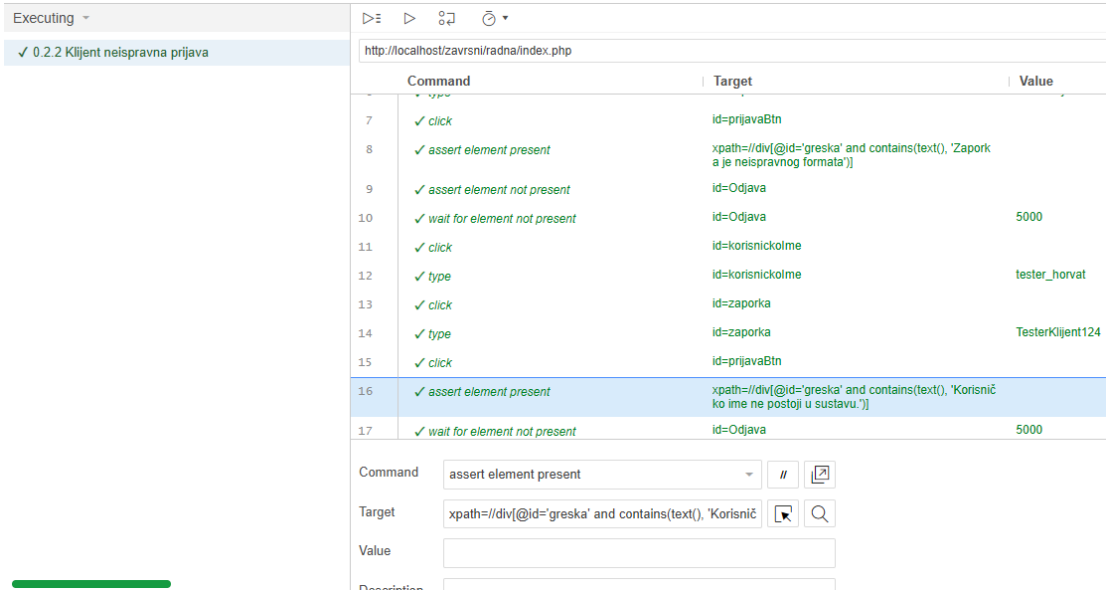


Slika 11: Testni scenariji izmjena profila, (Izvor: Snimka zaslona, 2025)

6.2.4.Testni scenarij neispravna prijava

Selenium IDE je korišten u izradi testa koji provjeri prikazuju li se poruke o grešci prikazano u *Slika 12*. U testu se unose neispravni podaci za korisničko ime i lozinku. Poruka o grešci se prikazuje na web stranici. Komande će provjeriti jesu li poruke prisutne u DOM-u. Ponovnim pokretanjem test će prikazivati poruke o grešci. Za razliku od prethodnih testiranja mora se pristupati sadržaju elementa. Pošto se koristi kombinacija elementa i sadržaja ovdje je odabrani *xpath* lokator. Štoviše korištenje konkretnog lokatora izbjegne se pogrešna interpretacija izraza od alata. Razlog specifikacije je u tome što se mora pronaći konkretna poruka *div* elementa.

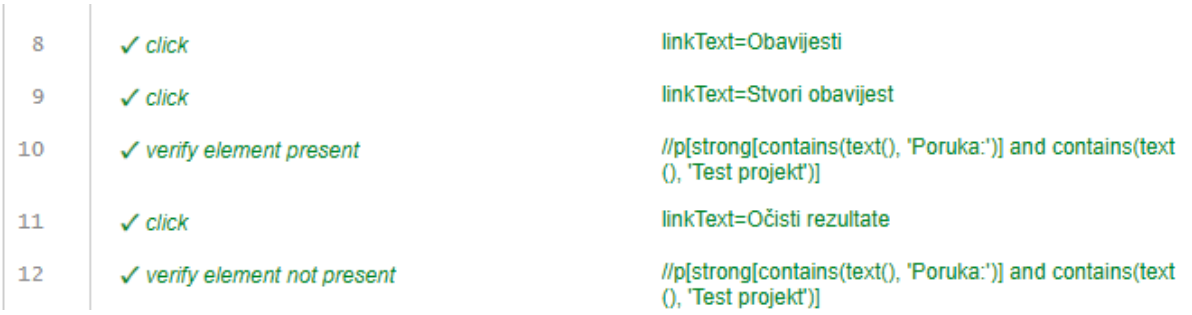
U komandama osam i šesnaest prikazane u *Slika 12* pronađu se poruke, a obim porukama je zajednički *div* elementa koji ima identifikator s vrijednošću *greska*. Osma komanda prolazi poruku čiji sadržaj glasi „Zaporka je neispravnog formata“. Šesnaesta komanda pronalazi element s poruku čiji sadržaj glasi „Korisničko ime ne postoji u sustavu“. Funkcija contains() u XPath-u koristi se kako bi provjerila sadrži li element tekstualni sadržaj koji odgovara dijelu zadanog argumenta. To omogućuje fleksibilniji odabir elemenata, posebno kada sadržaj može varirati ili sadržavati dodatne znakove.



Slika 12: Testni scenarij neispravna prijava, (Izvor: Snimka zaslona, 2025)

6.2.5.Testni scenarij klijent pregleda obavijesti

Prošli testovi bavili su se korisničkim testiranjem. Ovaj i daljnji testovi pripadaju funkcionalnim vrstama testiranja. Zbog razloga što se fokusiraju na jednu funkcionalnost uz što manju ovisnost o vizualnim elementima. Iz tog razloga su manji testovi koji nemaju velik broj ovisnosti i provjeravaju ispravnost jedne funkcionalnosti. Ovaj test će kroz pritiska na gumb pokrenuti skriptu koja stvara obavijest. Naravno kao u prošlim testovima uvjet ovog testa je prijava te se moraju izvršiti komande prijave u fazi pripreme. U fazu pripreme također pripada stvaranje obavijesti. Spomenute komande prijave i odjave su izostavljene iz *Slika 13* jer su spomenute u prošlim testiranjima. U osmoj komandi navigacija vodi klijenta na stranicu obavijesti. Nakon čega se stvori obavijest pritiskom na gumb. Deseta komanda provjerava da postoji element čiji tekst sadrži „Test projekt“. Time se želi provjeriti ako je obavijest stvarno stvorena. Rezultati se brišu u jedanaestoj naredbi. Dvanaesta komanda provjeri da ne postoji obavijest koja sadrži tekst „Test projekt“.



Slika 13: Testni scenarij pregled obavijesti, (Izvor: Snimka zaslona, 2025)

6.2.6. Testni scenarij klijent pregleda dodijeljene projekte i zadatke

Faza priprema podataka je istaknuta u ovom testiranju. Kao u prošli testovima komande prijave su izostavljene iz Slika 14. Osma komanda preusmjerava korisnika na web stranicu projekti. Deveta komanda stvara projekte i zadatke pritiskom na gumb. Deseta komanda provjerava postoji li div element. Dodatno se provjerava da li div sadrži dijete p element sa sadržajem „Test projekt“. Dodatno se provjerava drugo dijete img element s klasom img-fluid. Ova izraz će osigurati postoji li element za navedeni izraz. Sljedeća komanda preusmjerava klijenta u detalje testnog projekta klikom na sliku testnog projekta. Komanda dvanaest provjerava je li element učitan u DOM. Izraz komande pronalazi h5 koji sadrži strong element čiji je naziv „Naziv“. Za ostatak sadržaja h5 elementa provjerava se sadrži li „Test zadatak: Priprema terena“. Trinaesta komanda slična je dvanaestoj samo što se provjeri ako ostatak h5 elementa sadrži „Test zadatak: osiguranje dodatne opreme“. Navedeni projekt ima dva zadatka s javnom vidljivošću i jedan s privatnom vidljivošću. U četrnaestoj komandi provjerava se ne postoji li zadatak čiji sadržaj sadrži „Test zadatak: radovi čišćenja“.

U fazi čišćenja projekti se brišu u naredbom šesnaest klikom na gumb čiji je tekst „Očisti rezultate“. Uz brisanje projekta automatski se obrišu zadaci, zbog čega nije potrebno zasebno brisanje zadataka. Nakon brisanja, komanda sedamnaest provjerava da ne postoji testni projekt opisan u komandi deset. Posljednja komanda je pritisak na stavku navigacije „odjavaBtn“. Uspješno provođeni test osigurava da korisnici mogu pregledati izrađene projekte. Također je osigurano da se zadaci ispravno prikazuju korisniku, tj. da su vidljivi samo javni zadaci, dok privatni ostaju skriveni.

	Command	Target	Value
7	✓ click	id=prijavaBtn	
8	✓ click	linkText=Projekti	
9	✓ click	linkText=Stvori projekte i zadatke	
10	✓ verify element present	xpath=//div[contains(@class, 'container') and .//p[contains(text(), 'Test projekt')]]//img[@class='img-fluid']	
11	✓ click	xpath=//div[contains(@class, 'container') and .//p[contains(text(), 'Test projekt')]]//img[@class='img-fluid']	
12	✓ verify element present	//h5[strong[contains(text(), 'Naziv:')] and contains(text(), 'Test zadatak: Priprema terena')]	
13	✓ verify element present	//h5[strong[contains(text(), 'Naziv:')] and contains(text(), 'Test zadatak: osiguranje dodatne oprema')]	
14	✓ verify element not present	//h5[strong[contains(text(), 'Naziv:')] and contains(text(), 'Test zadatak: Radovi čišćenja')]	
15	✓ click	linkText=Projekti	
16	✓ click	linkText=Očisti rezultate	
17	✓ verify element not present	xpath=//div[contains(@class, 'container') and .//p[contains(text(), 'Test projekt')]]//img[@class='img-fluid']	
18	✓ click	id=odjavaBtn	

Slika 14: Testni scenarij pregled projekta i zadatka, (Izvor: Snimka zaslona, 2025)

6.2.7. Testni scenarij administrator odbaci zahtjev

U ovom testu na *Slika 15* postoje dvije ovisnosti koje se moraju pripremiti pored prijave. Prvo, administrator mora biti prijavljen i odabrati poveznicu „Zahtjevi“. Prva ovisnost koja omogućuje administratoru da može odbaciti zahtjev je izrada profila. Druga ovisnost odnosi se na poslani zahtjev od korisnika za izradu projekta. Nadalje da bi zahtjev mogao biti izrađen mora, prvo postojati profil. Razlog ovisnosti zahtjeva o profilu je taj što zahtjev ima kao vanjski ključ korisnika. Preko profila se može na učinkovit način povezati korisnik i zahtjev. Naime, zahtjev se prikazuje s nazivom, opisom, i imenom i prezimenom korisnika koji ga je poslao. Ime i prezime atributi su entiteta profil, te stoga mora biti najprije izrađen profil.

Izrada profila i zahtjeva provodi se klikom na gumbe „Izradi profil“ i „Stvori zahtjev“ u naredbama devet i dvanaest. Funkcionalnost koja se testira u naredbi trinaest pokreće se pritiskom na gumb „Odbij“. Funkcionalnost je zadužena za postavljanje statusa zahtjeva u status „Odgođeno“. Nakon pritiska gumba, status više neće biti vidljiv administratoru, što se provjerava u naredbi četrnaest. U naredbi šesnaest obriše se zahtjev. Profil se briše pritiskom na gumb u naredbi devetnaest. Posljednja naredba, dvadeset i dva služi kao dodatna provjera koja pokušava pronaći „Test projekt“.

9	✓ *click*	linkText=1. Stvori profil	
10	✓ *wait for element present*	linkText=Zahtjevi	30000
11	✓ *click*	linkText=Zahtjevi	
12	✓ *click*	linkText=2. Stvori zahtjev	
13	✓ *click*	linkText=Odbij	
14	✓ *verify element not present*	xpath=//div[@class='card-body']//p[contains(., 'Test projekt: Izgradnja obiteljske kuće')]	
15	✓ *click*	linkText=Zahtjevi	
16	✓ *click*	linkText=Obriši zahtjev	
17	✓ *wait for element present*	linkText=Zahtjevi	30000
18	✓ *click*	linkText=Zahtjevi	
19	✓ *click*	linkText=Obriši profil	
20	✓ *click*	linkText=Zahtjevi	
21	✓ *click*	linkText=Projekti	
22	✓ *verify element not present*	xpath=//div[contains(@class, 'container') and .//p[contains(text(), 'Test projekt')]]//img[@class='img-fluid']	

Slika 15: Testni scenarij odbaci zahtjev, (Izvor: Snimka zaslona, 2025)

6.2.8. Testni scenarij administrator izmjenjuje zadatak

Test administrator izmjenjuje zadatak u Slika 16, prvo izvršavajući naredbe prijave. Nadalje, naredbe otvaraju poveznicu koja vodi administratora do projekta. Naredba deset pritisne gumb „Stvori projekte i zadatke“ čime se izrađuje projekt sa zadatkom kao u prošlim testiranjima. Naredba jedanaest provjerava da testni projekt ne postoji u DOM-u. Naredba dvanaest klikanjem na sliku otvara detalje o projektu. U naredbama od trinaest do petnaest provjerava postoje li zadaci projekta na temelju naziva zadatka. U šesnaestoj naredbi također se provjerava prisutnost gumba „Promijeni“ u DOM-u. Naredba sedamnaest pokreće funkciju promjene zadatka klikom na gumb. Naredbama od osamnaest do dvadeset i pet izvršava se izmjena zadatka. Dvadeset i šesta naredba otvara novokreirani projekt s izmijenjenim zadacima. Nadalje, naredba dvadeset i sedam provjerava postojanje gumba za brisanje. U sljedećoj naredbi obriše se zadatak pritiskom gumba „Obriši“ za brisanje. Naredba dvadeset i devet ponovno otvara projekt i zadatke. Naredbe trideset provjerava je li naziv zadatka stvarno izmijenjen. U naredbi trideset i dva očiste se rezultati, što povlači brisanje projekta i pripadnih zadataka. Dodatno, se u naredbi trideset i tri provjerava da testni projekt ne postoji.

	Command	Target	Value
10	✓ click	linkText=Stvori projekte i zadatke	
11	✓ verify element present	xpath=//div[contains(@class, 'container') and .//p[contains(text(), 'Test projekt')]]//img[@class='img-fluid']	
12	✓ click	xpath=//div[contains(@class, 'container') and .//p[contains(text(), 'Test projekt')]]//img[@class='img-fluid']	
13	✓ verify element present	//h5[strong[contains(text(), 'Naziv:')] and contains(text(), 'Test zadatak: Priprema terena')]	
14	✓ verify element present	//h5[strong[contains(text(), 'Naziv:')] and contains(text(), 'Test zadatak: osiguranje dodatne oprema')]	
15	✓ verify element present	//h5[strong[contains(text(), 'Naziv:')] and contains(text(), 'Test zadatak: Radovi čišćenja')]	
16	✓ verify element present	(//div[@class='recenzija']//a[contains(@class, 'btn-primary') and text()='Promijeni'])[2]	
17	✓ click	(//div[@class='recenzija']//a[contains(@class, 'btn-primary') and text()='Promijeni'])[1]	
18	✓ click	id=IDStatusZadatak	
19	✓ select	id=IDStatusZadatak	label=Dovršeno
20	✓ click	id=IDVidljivost	
21	✓ click	id=IDVidljivost	
22	✓ select	id=IDVidljivost	label=Javno
23	✓ click	id=naziv	
24	✓ type	id=naziv	Test zadatak: Priprema okoline
25	✓ click	id=submit	
26	✓ click	xpath=//div[contains(@class, 'container') and .//p[contains(text(), 'Test projekt')]]//img[@class='img-fluid']	
27	✓ verify element present	(//div[@class='recenzija']//a[contains(@class, 'btn-danger') and text()='Obriši'])[2]	
28	✓ click	(//div[@class='recenzija']//a[contains(@class, 'btn-danger') and text()='Obriši'])[2]	
29	✓ click	xpath=//div[contains(@class, 'container') and .//p[contains(text(), 'Test projekt')]]//img[@class='img-fluid']	
30	✓ verify element present	//h5[strong[contains(text(), 'Naziv:')] and contains(text(), 'Test zadatak: Priprema okoline')]	
31	✓ click	linkText=Projekti	
32	✓ click	linkText=Očisti rezultate	
33	✓ verify element not present	xpath=//div[contains(@class, 'container') and .//p[contains(text(), 'Test projekt')]]//img[@class='img-fluid']	

Slika 16: Testni scenarij izmjena zadatka, (Izvor: Snimka zaslona, 2025)

6.3. Rezultati i daljnja unaprjeđenja

Testni scenariji spomenuti u radu obuhvaćaju ključne aspekte web aplikacije Gradi11. Rezultati testiranja dokazuju da web aplikacija ispravno izvršava osnovne funkcionalnosti prijave, registracije i manipulacije podacima. Ponovno provođenje testiranja za izmijenjene funkcionalnosti poželjno je jer osigurava da promjene budu ispravne na temelju uspješnih rezultata testiranja. Naravno, web aplikacija se može proširiti dodatnim funkcionalnostima. Gradi11 objektno strukturiran, što bi trebalo olakšati implementaciju novih funkcionalnost čime i povećati skalabilnosti. Prilikom implementacije novih funkcionalnosti, testovi za prijašnje funkcionalnosti moraju biti uspješni kako nove ne bi narušile starije funkcije. Isto tako, nove funkcionalnosti moraju biti pojedinačno testirane kako bi se osigura njihov ispravan rad.

Tijekom rada s alatom Selenium IDE zapaženi su prednosti i nedostaci njegova korištenja. Prije svega, svi testovi su spremljeni u jedan projekt, čime se postiže bolja organizacija. Projekt se može spremiti i ponovno uvesti u alat. Vrlo pozitivan aspekt testiranja u alatu je vizualno sučelje koje olakšava izradu i izmjene testova. Selenium kroz snimanje prati akcije poduzete na web aplikaciji i bilježi ih kao naredbe. Time se smanjuje potreba pisanja kôda za akcije. Štoviše, provode se male izmjene cilja i vrijednosti naredbe. Rijetko kad je potrebno izraditi vlastiti kod kao što je bio slučaj u testu izmjene profila. Alat sadrži ugrađene funkcionalnosti koje olakšavaju provođenje testiranja i uklanjanje grešaka. Jedna korisna funkcionalnost je *breakpoint*, koji omogućuje određivanje naredbe do koje Selenium pauzira izvršavanje testa. Uklanjanje greški vrlo je intuitivno jer *Log* označava naredbu u kojoj se greška manifestira. Uklanjanje greški je olakšano funkcijama *Play from here* i *Play to this point*. Naime, ako test padne, moguće je izvršiti izmjene naredbi i ponovno pokrenuti kritične dijelove testa.

Jedan od nedostataka alata je ovisnost o hijerarhiji elemenata i/ili identifikatorima. Naime, kako bi naredba, primjerice pritiska gumba bila uspješna, alat mora ciljat ispravan element. Često se događa da web aplikacija prolazi kroz redizajn i implementaciju novih dizajnerskih trendova. To može uzrokovati sukob za naredbe koje ovise o elementima koji više ne postoje u aplikaciji. Drugi nedostatak je ovisnost brzine izvođenja testa o učitavanju elementa. Najveća brzina izvođenja testa može spriječiti dovoljno dugo vrijeme za učitanje elementa i rezultirati padom testa. Zbog čega je potrebno smanjiti brzinu izvođenja testiranja. Posljednje ograničenja alata odnosi se na snimanje uvoza datoteke, zbog čega je potrebno izraditi skriptu koja uvozi prazne datoteke, primjerice za testiranje izmjeni profil.

Testiranje u Selenium IDE osiguralo je da web aplikacija Gradi11 ispravno izvršava svoje funkcije. Testiranja nisu samo potvrdila ispravno funkcioniranje web aplikacije, već su ukazala na moguća unaprjeđenja.

7. Zaključak

Testiranja su dio životnog ciklusa izrade web aplikacije. Provođenjem testiranja osigura se ispravno funkcioniranje aplikacije i njezinih funkcionalnosti. Aplikacija s izrađenim testovima bit će lakše za održavanje. Promjene će se regresijskim testiranjima brže provoditi jer će greška biti prije uočena. Nove funkcionalnosti će biti implementirane brže jer će testovi pokazati kako utječu na postojeće funkcije i ugrožavaju li ih. Odabir automatiziranog testiranja umjesto ručnog testiranja je bolje rješenje. Jednom kreirani, isti testovi mogu se izvoditi više puta, čime se smanjuje ručno ponavljanje testova. Budući da se testiranje provodi alat, smanjuje se mogućnost pojave grešaka uzrokovane ljudskim faktorom. Za projekte koji se mogu automatizirati, bolje rezultate postići će automatizirano testiranjem zbog manjeg utroška resursa i vremena.

Rano testiranje od samog početka izrade web aplikacije osigurava stabilnost i skalabilnost. Izrađene funkcije funkcioniraju prema zadanim zahtjevima. Dodatne funkcije zadane od klijenta mogu biti implementirane sigurno s ciljem razvijanja složenijeg rješenja. Iako u početnoj fazi zahtijevaju veći vremenski ulog i potencijalnu suradnju testnog tima, za veće web aplikacije u konačnici smanjuju greške i time ubrzavaju razvojni proces. Testiranje može biti provođeno od iste osobe koja implementira rješenje za manje razvojne projekte. Za veće projekte formira se testni tim koji provodi nepristrano testiranje web aplikacije Za web aplikaciju Gradi11 provedena su testiranja i osigurano je da funkcije izvršavaju namjenu. Uspješni testni scenariji potvrđuju ispravnost funkcionalnosti. Pouzdanost u web aplikaciju poveća se korištenjem više vrsta testiranja spomenutih u radu. Uvođenjem testiranja u razvojni proces osigurava se kvalitetnije programsko rješenje.

Popis literature

[1] J. Robbins, *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics*, 4th edition. Beijing: O'Reilly Media, 2012.

[2] D. Stuttard and M. Pinto, *The web application hacker's handbook: finding and exploiting security flaws*, 2nd ed. Indianapolis, IN : Chichester: Wiley ; John Wiley [distributor], 2011.

[3] I. Sommerville, *Software Engineering, 9/e*. London, Ujedinjeno Kraljevstvo: Dorling Kindersley, 2011.

[4] L. Shklar and R. Rosen, *Web Application Architecture: Principles, Protocols and Practices*. Wiley, 2003.

[5] A. Mozaffar, *Mastering Blazor WebAssembly: A step-by-step guide to developing advanced single-page applications with Blazor WebAssembly*. Packt Publishing Ltd, 2023.

[6] M. O. A. Course, *Software Development Fundamentals: Exam 98-361 MTA*, 1st edition. Wiley, 2011.

[7] "Computer programming language | Types & Examples | Britannica." Pristupljeno: Aug. 08, 2024. [Na Mreži]. Dostupno na: https://www.britannica.com/technology/computer-programming-language

[8] T. Felke-Morris, *Web Development and Design Foundations with HTML5*, 10th edition. Pearson, 2020.

[9] J. Lam, I. Alexander, and M. Kmiec, *Raggett on Html 4*, 2nd edition. Harlow, England ; Reading, Mass: Addison-Wesley, 1997.

[10] C. Musciano and B. Kennedy, *HTML and XHTML, the Definitive Guide*. O'Reilly, 2000.

[11] J. Duckett, *HTML and CSS: Design and Build Websites*. John Wiley & Sons, 2011.

[12] J. Duckett, *PHP & MySQL: Server-side Web Development*. Wiley, 2022.

[13] O. Stax, *Introduction to Python Programming*. Open Stax Textbooks, 2024.

[14] J. Albahari, *C# 10 in a Nutshell: The Definitive Reference*. O'Reilly Media, Inc., 2022.

[15] A. Starkov, *Learning .NET MAUI: Unlock the potential of .NET MAUI for Cross-Platform app development (English Edition)*. BPB Publications, 2023.

[16] B. H. David Gelperin, "The growth of software testing," *Commun. ACM*, vol. 31, pp. 687–695, Jun. 1988, doi: 10.1145/62959.62965.

[17] G. J. Myers, T. Badgett, and C. Sandler, *The art of software testing: now covers testing for usability, smartphone apps, and agile development environments*, 3. ed. Hoboken, NJ: Wiley, 2012.

[18] D. Graham, *Foundations of Software Testing: ISTQB Certification*. Course Technology Cengage Learning, 2008.

[19] P. Jorgensen, *Software testing: a craftman's approach*, Fourth edition. Boca Raton: Auerbach, 2018.

[20] J. Preece, Y. Rogers, and H. Sharp, *Interaction design: beyond human-computer interaction*, Fourth edition. Chichester: Wiley, 2015.

[21] M. Ehmer and F. Khan, "A Comparative Study of White Box, Black Box and Grey Box Testing Techniques," *Int. J. Adv. Comput. Sci. Appl.*, vol. 3, no. 6, 2012, doi: 10.14569/IJACSA.2012.030603.

[22] G. Meszaros, *xUnit Test Patterns: Refactoring Test Code*. Pearson Education, 2007.

[23] R. Black, *Managing the Testing Process*. New Jersey: John Wiley & Sons, 2002.

[24] G. D. Everett and R. McLeod, *Software testing: testing across the entire software development life cycle*. Piscataway, New Jersey: IEEE Press, 2015.

[25] D. KONDRATIUK, *UI TESTING WITH PUPPETEER implement end-to-end testing and browser automation using javascript... and node.js*. S.l.: PACKT PUBLISHING LIMITED, 2021.
[26] L. Gelfenbuim, *Web Testing with Cypress: Run End-to-End Tests, Integration Tests, Unit Tests Across Web Apps, Browsers and Cross-Platforms*. Place of publication not identified: BPB Publications, 2022.
[27] R. Gupta, *Ultimate Selenium WebDriver for Test Automation Build and Implement Automated Web Testing Frameworks Using Java, Selenium WebDriver and Selenium Grid for e-Commerce, Healthcare, Edtech, Banking, and SAAS (English Edition)*. Delhi: Orange Education PVT Ltd, 2024.
[28] "Results of the first ever selenium survey," Selenium. Pristupljeno: Aug. 26, 2024. [Na Mreži]. Dostupno na: https://www.selenium.dev/blog/2021/selenium-survey-results/

Popis slika

Popis tablica

Prilog

Ovom radu priložen je programski kod aplikacije i datoteke testova iz alata Selenium IDE u zip datoteci koja je dostupna u sustavu radovi.foi.hr.